InCloud - Towards Infotainment Services For VANETs

by

Haolin Guo

Thesis submitted to the
Faculty of Graduate and Postdoctoral Studies
In partial fulfillment of the requirements
For the M.A.Sc. degree in
Electrical and Computer Engineering

School of Electrical Engineering and Computer Science
Faculty of Engineering
University of Ottawa

Abstract

In order to realize effective infotainment systems for vehicles, we need to have context-aware applications that use the latest (live) information for an enhanced user experience. Such up-to-date information is now abundantly available on the Internet, due to the explosive growth of the Web 2.0. In earlier times, it was difficult and expensive for vehicles to connect to the Internet. Recent advances in vehicular ad-hoc networks (VANETs) have enabled vehicles to connect to the Internet through road side infrastructures, with little to no additional cost. However, there are several problems with directly using Internet data in a vehicle, such as (1) Internet data sources have their own interfaces, which keep changing over time, needing frequent application updates, (2) information provided by multiple data sources needs to be preprocessed and fused before use, and (3) vehicles employ propriety platforms for infotainment systems, which makes an application update even more cumbersome. Furthermore, accessing multiple Internet sources may cause unnecessary overhead over the VANET bandwidth. In this thesis, we propose a cloud-based middleware framework for vehicular infotainment application development. The proposed framework follows service oriented architecture in which data filtering and fusion functionalities are delegated to the cloud. Data filtering and fusion reduce the data flow over VANET. Furthermore, because most the the processing is done on the cloud, the client becomes lightweight and loosely coupled with Internet resources and underlying platforms. We also propose a class-based fusion method to combine information from multiple sources. The efficacy of the proposed framework is validated by developing an enhanced navigation (eDirection) application for the vehicle, as well as three infotainment applications: context-aware music, news, and weather.

Acknowledgements

To my family: Wenli Guo(My father), **Jinjuan Geng**(My mother). I will not be able to study in Canada without their supports"

Special Thanks to:

Prof. Dr. Abdulmotaleb El Saddik, my supervisor. I am very lucky to study in his lab. He gave constructive advises on crucial questions of my research.

Dr.Mukesh Saini, my co-supervisor. He taught me a lot about my research. At the same time, he is also a very kind, easygoing person and a good friend.

Dr.Dewan Tanvir Ahmed, my co-supervisor. I learned many things from him through our first paper.

All MCR labs members, I am very happy to study with them. They are all very kind and generous people.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Since the advent of the first automobile, various advanced technologies have been developed to improve the driving experience. Among these technologies, several increase driving safety such as Anti-lock Braking System (ABS), Traction Control System (TCS); some provide better operability like cruise contro and all wheel drive; while others improve driving conditions such as anti-lock brakes and climate control. The development of Intelligent Transport Systems (ITS) [15] has further improved driving efficiency. Recently, a number of infotainment applications have been built for the comfort and convenience of the drivers and passengers. These new technologies help passengers enjoy their journey instead of just trying to pass the travel time.

The development of driving technologies is not confined to hardware, vehicular technologies in software are also becoming increasingly important, for example multimedia, navigation, and health monitors. Some infotainment applications are being applied in more and more vehicles for a robust operation and a higher degree of convenience. Since cloud computing has been accepted by network resource providers, more systems are being developed based on this technology. In this new development paradigm, the core application parts are delegated to the cloud server on the Internet. The local clients only maintain the user interface and display parts of the application. The emergence of Vehicular Ad-hoc Networks (VANETs) provides the possibility for the vehicles to access the Internet. Then, a great number of vehicles systems renew the model to adapt to Internet data. This thesis proposes a system model and implementation for an infotainment system. The model takes data from the Internet for providing more efficient services for vehicles. Also, we proposed an approach to establish system with low-requirement and re-usability advantages.

1.1 Motivation

Nowadays, there is hardly a vehicle that doesn't possess an infotainment system [12]. But what if a navigation system shows a wrong pathway because the map is not updated

timely? Or what if the music becomes boring because the playlist is not intelligent enough to adapt to current contexts and ratings? We need timely and accurate information to develop effective infotainment applications, which is fortunately available in digital format online on the Internet. In fact, most of the ITS still rely on the analysis of local data, which is limited and generally out of date; such data surely influences the quality of the outputs given by the system. Although the infotainment system is not critically necessary for primary driving tasks like the vehicle power system, its popularity still validates its existence. Since the information sources in the Internet are updated timely, we consider updating the infotainment systems with information from the Internet. Passengers can watch videos, listen to music, and even play games online, while, the driver can access personal email, news, etc. These infotainment applications can greatly improve the driving experience. For example, one will get bored after he or she listens to local songs for a long time. Generally, a song lasts three or four minutes, which means that even if a user chooses 30 of his favorite songs, these can only be played repeatedly for two hours. According to the Air Resources Board(ARB) survey 2009 [49], the average time people spend driving is 18 hours and 31 minutes a week. This includes about 2 hours 52 minutes a day on weekdays, and 2 hours 7 minutes a day on weekends. We can see that the audio resources are repeated even in one day's average driving time. Also, most people generally do not have time to search and update new audio resources, therefore it's very easy for the passengers and driver to get bored with repeated multimedia resources. The situation would be different if the vehicles could access the Internet; make use of user profile and context information and search for appropriate titles. Users could listen to music directly from online sources like Last.fm or get personalized media resource like rock music.

1.2 Research Issues

Our first research issue is that how can we get vehicles access to the Internet. Cellphone has gained the access to the Internet through MANET today. Due to the VANETs, vehicles can also use the Internet. Our system is built with the VANETs as communication infrastructure. We will talk about it in detail in next chapter. Therefore, vehicles could offer context-aware [8] cloud services to the driver and other on board passengers through VANETs just as the cellphones in the mobile network. Then we need to develop a system to process these data. To make our infotainment system more efficient and reliable, we bring several features into the establishment of our model, like a simple interface, easy operation and context-awareness. However, we still need to overcome a set of challenges to efficiently use Internet resources, as follows:

1. Most information resources have their own application programming interface (APIs) for accessing the data. The resource owners update these APIs over time, according to their own requirements. In order to make the infotainment applications robust to

these changes, our system should be loose coupling, which means that each function in our application is independent.

2. Since the connection of vehicles under the VANETs is wireless. The limitation of bandwidth has restricted data transmission. Also, the stability of the network affects the efficiency of infotainment system. Abundant data will bring the transmission delay and the data loss to the vehicles which is deadly to time critical data.

3. Standard APIs return information with a lot of redundant data which is not useful for infotainment applications. The redundant data reduces the system's efficiency and complicates data format, which is not suitable for data fusion. Therefore, we need to filter the data into a unified format.

4. Vehicle manufacturers build their own infotainment systems by cooperating with different IT companies. For example, Apple supports BWM-iDrive and Microsoft supports Ford MyFord [27]. These companies all have their own development platforms. There is no standards for developing a cross-platform infotainment system. Our goal is to develop a system that can execute in most of platforms.

5. There are a number of sources that provide similar information. For example, we can obtain weather information or music ratings from the cloud resources. The problems are determining which resource we should trust more, and how to integrate information from multiple sources.

6. The limited interactive interface and driving environment restricts users' operations. The operations can't be complex and prolonged. For example, the driver can not input complex commands like typing destination names into the system while driving. For this reason, we need to provide efficient services with a concise and clear interface.

In this thesis, we propose a solution to the above problems. We build a middleware cloud server under the backbone of the VANETs. We take service oriented architecture (SOA) design approach for our implementation. The system in the server can filter the redundant data from the Internet and occupy most of data processing work. In this way, it may reduce the consumption of network bandwidth and hardware requirement of the infotainment client in the vehicles. We will discuss this in greater detail in following chapters.

1.3 Thesis Contributions

The main contributions of this thesis are the following:

- Design data fusion and context-aware pre-filtering strategy for infotainment applications, which can effectively combine information from multiple sources, therefore reducing the data flow on the RSU-OBU link.

- Design and implement an enhanced routing algorithm using data fusion strategy. The enhanced algorithm can offer optimized route which avoids the constructions from third party resources.
- Design and implement a context aware search algorithm for music searching. The proposed algorithm fuses the rating data from different music search resources with users music preference. Therefore, we can provide users with more reasonable music search results.
- Design and develop cloud services so that the applications are loosely coupled with the data resources and underlying platforms, which is a desired feature for adaptive and easily reconfigurable applications.

1.4 Thesis Overview

The organization of this thesis is as follows:

- Chapter 1 is introduction. In this chapter, we first talk about the research motivation and research issues. We also state the research contributions.
- Chapter 2 is literature review. It first introduces VANETs, Infotainment System History and Service-Oriented Architecture (SOA). Then, it shows the design concept of mashup which is extend of SOA. Last, it reviews the architecture similar to our work and compare the differences between our work and some representative work.
- Chapter 3 is proposed work. We firstly give out the system architecture and design principles. Then we elaborate on the architecture in the following dissertation.
- Chapters 4 demonstrates the implementation of our proposed system. We explain four applicatoins deisgn and fucntionalites. We also explain the details of the system design and at the same time clarify how the prototypical system reflects the features of SOA.
- Chapter 5 provides details on the experiments that allowed us to verify the feasibility and efficiency of our system.
- Chapter 6 finally concludes the thesis and suggests some future work and improvements.

Chapter 2

Literature Review

We live in a fast-paced society where the timely and accurate sharing of information among people can make our life more convenient and comfortable. The Internet provides a distributed platform for information exchange. The Web 2.0 [33] technology takes the information spread to a new level. With the help of the Web 2.0 technology, many web services are developed rapidly in order to satisfy users' needs. More and more companies and software firms tend to offer their services through the web. Instead of selling software to customers offline, these companies offer customers an updated service through web services. Users or developers can pick useful cloud services to form new applications, and since plenty of sources provide different kinds of data and without any defined standards to follow, it is a challenge to collect relevant data from different sources on the Internet and to process that information efficiently, according to user's needs. In this thesis, we design a mashup system [1] based on Service Oriented Architecture (SOA) for VANETs [39] which provides context-aware services in real-time. This design enables the system to retrieve data from the Internet with a certain amount of flexibility and process them according to defined logic. SOA is currently the most commonly used approach to realize a Cloud based system, but the traditional SOA implementing method still has disadvantages. In this thesis, we propose to development an approach that can overcome these disadvantages. [44]. Our service model enhances the in-car infotainment system. In this chapter, we will introduce VANETs and infotainment system; and provide a brief overview of the SOA and Mashup design paradigms in order to build background knowledge for the thesis work.

2.1 VANETs

Wireless communication technologies emerged for more than 100 years ago, but have evolved a great deal over the years. In 1879, David E. Hughes created a clockwork keyed transmitter and now, the wireless technologies have become an indispensable part of the

communication approach. However, it still has drawbacks, such as a low and unstable bandwidth, limitations of data transmission range, etc. Recent advances in wireless networks have introduced a new type of network called the Vehicular Ad-hoc Networks (VANETs), which enables Internet access in vehicles on the move. VANETs facilitate inter-vehicle communication between nearby vehicles and/or allows communication between vehicles and roadside infrastructures [20].Vehicular Ad-Hoc Networks (VANETs) is one of the system's many sub-classes. The others are Mobile Ad-Hoc Networks (MANETs), Wireless Sensor Networks (WSNs) and Wireless Mesh Networks (WMNs) [19]. VANETs add a new dimension when accessing Internet services. The fast growing Internet offers plenty of services that are accessible anytime and anywhere. Our roadway travel could be more interesting and convenient if infotainment systems could make proper use of the resources available on the Internet in order to ensure infotainment applications are up-to-date and effective.

A VANET, as shown in figure 2.1, is a collection of moving vehicles and road side infrastructures. The network disseminates information through these network nodes in a manner similar to cells mobile in the Mobile Ad-Hoc Networks (MANETs). Communication components on the vehicles are called on-board units (OBU) [46] and consist of wireless modules and sensors. The sensors are used to monitor and transmit the vehicles information, and the Road Side Unit (RSU) [46] forwards information from one OBU to the other. RSU not only provides the driving data such as location, speed, heading, etc., but also supports Internet access for vehicles. Our system is created under the support of Internet access from the VANETs.

Although VANETs enable Internet access in vehicles, we still need to overcome a set of challenges to properly collect information from multiple sources on the Internet. First, data from multiple sources may consume a large amount of bandwidth between the road side unit (RSU) and the on-board unit (OBU) [20]. Since the link between the OBU and the RSU is intermittent, we need to minimize data flow over this link [11]. Second, the applications should be loosely coupled with the Internet source's specific APIs and the underlying hardware/software platform in the vehicle.

2.2 Infotainment System History

Infotainment is a combination of Information and Entertainment, which means information and entertainment served together. In the context of vehicles, an infotainment system can provide applications for navigation and music, in order to enhance the driver and/or passenger experience. Figure 2.2 shows the development history of the in-vehicle infotainment system. The first infotainment system was introduced in vehicles by Motorola in the form of a radio in 1930 [5]. This is the basis for the infotainment system. In subsequent years, auto manufacturers around the world accepted this idea that driving should be enjoyable. Chrysler introduced a record player in their cars, and in 1965, Ford released the first tape

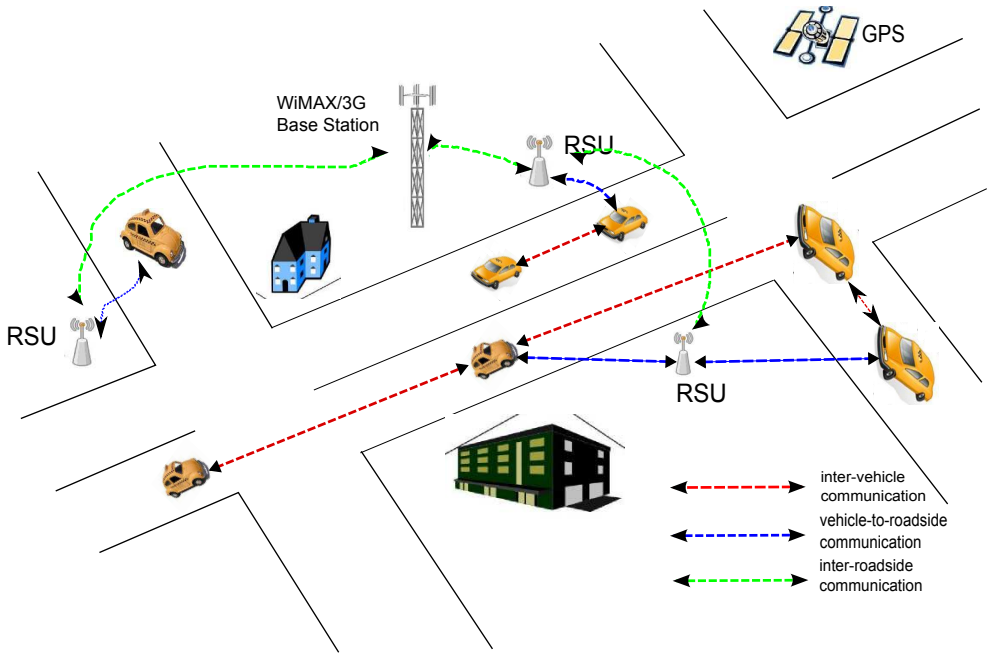


Figure 2.1: Schematic diagram of VANETs.

player. These innovations are followed by the introduction of cassette-tape players, stereo systems, and in-dash CD players. With the development of Bluetooth and hardware storage, the in-car infotainment system has integrated satellite radio, mobile devices, and local stored multimedia together.

The navigation application shows that the infotainment system can bring us something more than just entertainment. Before the invention of the navigation system, long road driving meant a bunch of fold-out maps. We also used to get route information from online services such as Google map, or Yahoo map, print the out the results and use them as the tour guide. The navigation system changes this situation, as it gives convenience to the users, allowings them to follow the route dynamically. The Global Positioning System (GPS) offers essential data for navigation. GPS was first used by the U.S. military in the 1960s [35], and is only available to the public since the early 1980s, and then, the data was not very precise. By the end of 2000, GPS navigation with precise data began to open to the public. There are currently 16 major original equipment manufacturers(OEMs) with 78 major brands and approximately 600 models [10]. Almost all of these models are equipped with infotainment systems. Future cars will certainly be able to communicate with each other through VANETs. In this section we provide background knowledge essential to our proposed system.

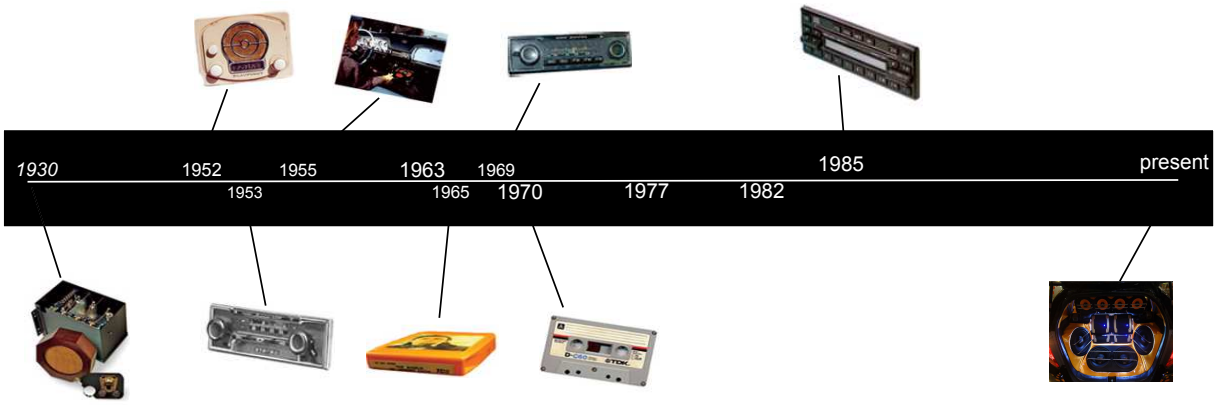


Figure 2.2: The history of in-vehicle infotainment system.

2.3 Service-Oriented Architecture(SOA)

SOA is a software design pattern that provides each application to function as an independent service. It is the main pattern used to implement cloud based system. SOA is defined by the OASIS group [6] as a paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains. In this pattern, one application is divided into N ($N > 0$) functional modules. Each module called service, is a self-contained unit that can hold one process to produce one individual functional task. For instance, the following are the services in the travel plan application: the service for mapping, the service for booking tickets, the service for searching the places of interests, and the service to calculate the cost. Each service can provide the user with different functions. These functions are associated and loosely coupled. Each service is responsible for one action or task, and the application organizes the services in the right processing order.

The aim of inducing SOA is to design large-scale, tightly integrated applications that are modular, inter-operable, and reusable. The services are encapsulated so that they can be invoked across the application interfaces. We also want the services to serve as objects. These objects are available for multi-connects so that they can save more time and money by reducing duplicated code work. Current infotainment systems follow an ad-hoc software development approach (monolithic architecture), which does not meet these requirements [34] [2] [24]. In this work, we propose a cloud-based middleware following Service-oriented architecture for infotainment over VANETs, adhering to the above issues. SOA has the potential of reducing both development costs and future application delivery time [47]. At the same time, the flexible and agile organization gives the developer real-time access, which means the upgrades and maintenance of the system in SOA are much easier and faster. The SOA has the following characteristics [3]: 1) Re-usability, 2)Loose coupling, 3) Comparability, 4) Statelessness. Currently, the common approach to realize SOA is to

implement it by web services. Services use WSDL to describe the service interface or an end point that operates the input and output parameters. Normally the result of web service contains the name of the web service interface. For the data, web service generally acquires XML or JSON data type. For the transmiting protocol, web service uses the XML-based protocol such as SOAP, WS-notification, and Representational State Transfer (REST) [48].

In this thesis, we propose a cloud-based middleware following SOA for infotainment over VANETs adhering to the above issues. The cloud services run on the RSU or other standard clouds such as Amazon. The proposed middleware performs most of the computation tasks in the cloud and presents only infotainment related content to the vehicular applications. As a result, the application client can be easily adapted to multiple platforms, while new features can be incorporated efficiently in the cloud. To reduce the data flow from RSU to OBU, we take following steps: (1) pre-process Internet data to remove content not related to infotainment applications, (2) filter Internet data according to the current context to further reduce its size, (3) fuse data from multiple Internet resources in the cloud itself so that we utilize information from multiple resources, but the data flow from RSU to OBU would still be equivalent to one resource. To demonstrate the efficacy of the proposed middleware system, we have implemented the proposed architecture and developed three lightweight applications (efficient up-to-date routing, context aware news, and music recommender).

2.4 Mashup

We also optimize the cloud based architecture by extending it with Mashup design. A mashup is a desgin that uses content from more than one source to create a single new service displayed in a single graphical interface [14]. Mashup is firstly known as a music album. A music bank mixed the Beatles' White Album and Jay-Z's black album together as a new album called Grey album. When brought to SOA field, it is a design that adds a middle ware between server and client. It is used in web development for combining and aggregating existing source data to generate more useful data. This part of design usually maintains the service composition and data integration. Theoretical SOA architecture has very low efficiency unless the web services are specifically designed for the applications. For example, if we directly use geographic data from Yahoo as keyword to do a routing search in Google Map API, it may return failed result. Because data format or data unit does not match. Then we need to unify these data in order to use them. Another, if we want to process the API feedback further with the specified requirements, we have to add one step between API feedback and result. The middleware server plays this role in the architecture. It adds the local database that can store personal information like users' behaviors and user private data.

2.5 Related Work

Infotainment systems play a significant role in making our travel comfortable and enjoyable. Many research attempts have been made to improve infotainment systems from various aspects. We review existing commercial infotainment systems and related research works. We particularly focus on in-vehicle infotainment systems. Currently, the main functions of infotainment systems are navigation and multimedia based entertainment. Most in-vehicle infotainment systems by automobile manufacturers still rely on locally stored data for these two functions. Although some tele-data is used, such as GPS data and radio data, there are still limitations. For example, for most vehicles, the map data of the navigation system is stored locally. A lot of live traffic data or road conditions data can therefore not be used by the system in a timely fashion. In addition, users do not want to pay for a whole new map, every time there are few small roads that are built. Besides, most vehicle manufactures tend to enlarge the resource scope of the system by adding a Bluetooth module. Since our mobile phone can access the Internet, vehicles can also get the Internet data through the Bluetooth connection. However, unstable bandwidth and limited speed can not satisfy the requirements of the next-generation of infotainment systems; there is still a great deal of room for improvement in the approach being used for data collection.

Another important problem is that there is no unified standard development environment for manufactures to use to develop the infotainment systems. Presently, companies mostly use proprietary platforms to develop their own infotainment systems. For developers, this means a longer application development time, since they need to set-up the specified environment and acquire its development tools. Although some automotive manufacturers and major IT companies have started to develop open source platforms for infotainment systems like GENAVA InVehicle Infortainment (GENIVI), these platforms are hard to use because of the lack of related standards [26].

In this thesis, we mainly focus on the application layer. We wish to set up architecture that implements the cloud based infotainment application. Although infotainment systems have been around for decades, the topic of cloud based infotainment systems is still new and emerging. Only recently have a reasonable and feasible architectures been proposed [31]. Among these proposed architectures, web services are the principal means of implementing the Internet access. This is a method that supports machine communication over the network. Generally, the communication is through Http or XMLHttp protocols. The data package is described using SOAP or REST, and exchanged through the interface that is defined in a machine-processable format like WSDL.

We researched the infotainment systems that invoke Internet sources using web services. Currently, there are two ways for the systems to use the Internet sources. One is directly invoking web services from the Internet, as described in [16]. In their prototypical implementation, they directly display the route calculation results from BingMaps web services. The other approach is to create a middleware server to customize services so that the server grabs the data from the Internet [29] [30]. Among these cases, most do not

Figure 2.3: Prototypical implementation of an in-car-infotainment system

use cloud, while others only provide theoretical models. We show several cases to make a detailed comparison:

A Prototypical Implementation by Daimler Center

Daimler Center for Automotive IT Innovations proposed a prototypical implementation of an intelligent in-car-infotainment system [16]. Their work is a context-aware, intelligent infotainment system. This system can adapt the service results based on the context-dependent daily routines of the user. They implemented the applications in ActionScript that can simulate the driving situation from the personal desktop. For the services, they invoked the BingMaps webs services for route calculation. for this system, Sandro proposed a contextual and personalized design for user interface, but they only implemented user's pattern recognition and context clustering the client side. There is no further talk of the context aware design for service feedback. Figure 2.3 is a screenshot of their system.

Implementation of the Android-Based Automotive Infortainment System for Supporting Drivers Safe Driving

While some researches on automotive infotainment systems focus on the users' convenience, some similar works focus on the driver safety. For example, Minyoung Kim and his team proposed a design with additional functions such as black box and self-diagnosis, which they added to the automotive infotainment systems, based on the android platform embedded hardware [26]. The highlight of their work is the set up of an architecture using an Android

Figure 2.4: Implementation of the Android-based automotive infotainment system for supporting drivers safety

platform that is independent from current automotive development platforms like GENIVI. We can have a larger set of applications to choose from, since Android Market can provide various applications. The open source platform allows the addition of adaptive third-party applications based in user preferences.

An In-Vehicle Infotainment Software Architecture Based on Google Android

Gianpaolo Macario also proposed an Android based architecture for in-vehicle infotainment [31]. Similar to Minyoung's work, they proposed an architecture that can attach third-party applications based on the Android platform. In this work, they proposed a safe mechanism for third-party applications to access vehicle data. For instance, they granted permission to applications to gather the temperature and speed values; they also used Google Android features to handle inter-application communication. To verify the feasibility, they implemented a prototype system using this architecture. Unfortunately, they did not provide related experiment results and their architecture doesn't adopt the cloud technology.

Integrated Embedded System Architecture for In-vehicle Infotainment

National Chia-Yi University developed a system called Mayday, which provided location-based services for vehicles through Wifi and (General Packet Radio Service)GPRS [21]. GPRS is a packet oriented mobile data service on the 2G and 3G cellular communication system's global system for mobile communications [7]. To meet the need for vehicle the Internet access from anywhere, a distributed service-based architecture was proposed based on Jini middleware technology. They proposed an embedded system architecture, as shown in figure 2.5, which consists of the service and control station segment (SCS), and the mobile segment (MS). This system took the location information from a GPS receiver. For

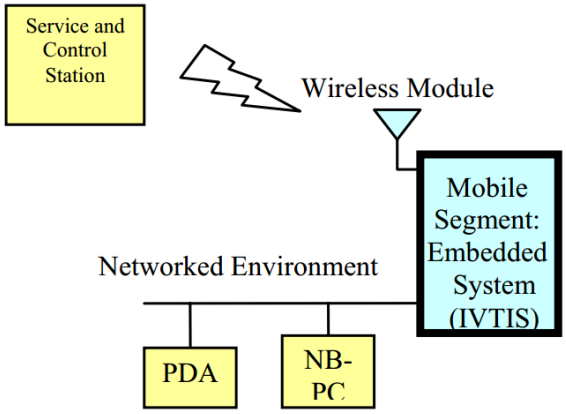


Figure 2.5: Proposed embedded system architecture

infotainment purposes, the system had an in-vehicle network that worked as a LAN server to provide a connection to stream data with other devices such as PC, PDA, and tablet. However, this paper didn't mention the middleware server and customized services model. The middleware server plays an important role in telematics.

Other In-vehicle Infotainment related works

Besides these principal related works, we also skimmed through other infotainment works. To make the point clear, we created a Table 2.1, which compares these previous works to our proposed work.We mainly consider these related work from two aspects. One aspect is efficiency and versatility of the infotainment system. Hackbarth [18] proposed a modular architecture for infotainment systems, mainly designed to meet the needs of various hardware and configurations. Macario et al. [31] argue for am Android platform for infotainment system development, to make it accessible to most users. Al-Ani [2] also proposed an android-based infotainment system. While Android provides a generic platform for infotainment applications, car manufacturers still prefers their own platform because of specific requirements and features.

Another important aspect considered by researchers is the networking protocols of VANETs. Festag et al. [13] proposed a geocast routing protocol customized for safety, efficiency, and infotainment messages, Huang et al. [22] also proposed a routing mechanism for infotainment messages, Cheng et al. [9] proposed a link layer optimization for infotainment messages. Also, Salvo et al. [38] [36] provide forwarding rules so that infotainment messages reach maximum nodes, and Amadeo et al. [4] propose an enhancement of IEEE 802.11p to better accommodate the traffic information that is generated through the infotainment application.

Researchers have proposed several techniques to improve the user interface of infotainment systems. Sharma et al. [40] [41] proposed a component-based human-machine interface framework that adapts the interface of infotainment systems according to vehicular context. Different ways of connecting external devices to car infotainment systems have also been explored [42] [43]. Ohn-Bar et al. [32] proposed a vision-based control for infotainment systems. where the driver and the passengers can interact with the system with gestures rather than with a tactile interface.

To cope with the processing needs of infotainment applications, Hsu et al. [21] and Lattanzi et al. [28] proposed the use of high power embedded processors. We argue that such a solution would not be scalable. Also, processing Internet data in the cloud would reduce the bandwidth requirements, as only the results would be sent to the infotainment system over VANETs. Recently, Hussain et al. [23] [24] briefly discussed the idea of using cloud over VANETs, but no framework is proposed. The authors merely discuss the idea that the vehicle can use RSU as a gateway to the Internet cloud, but there needs to be a detailed investigation of the approach. Kim et al. [25] delegates a malware detection process to the cloud in infotainment systems. Rangarajan et al. [34] discuss security on cloud-based infotainment.

A comparison of the proposed work with previous works is provided in Table 2.1, based on the following aspects: Network or VANETs considered or not? Cloud used or not? Internet resources fused or not? If includes a data filtering step or not? And have the authors identified cloud services for current infotainment systems or not? In this work, we propose a cloud-based infotainment framework for VANETs and explore the design issues of such a framework. We further validate our framework by building three infotainment applications based on the proposed framework where we consider resource information adaptors, data unification, data fusion, and service identification.

With respect to the previous literature, our goals are to:

1. Use new approaches to collect Internet data.
2. Design a data fusion strategy.
3. Implement the proposed architecture and validate it through experiments.

We determined that it would be beneficial to our work to incorporate the features of the customized services model into our model. One advantage is that our system works for on-demand data integration which can reduce data size by dropping the redundant data. Another advantage is improved system efficiency; since we move the data processing part and services model to the server, it reduces the delay and reaction times. It reduces hardware requirement on the client side and is therefore for manufacturers to adopt. Moreover, we designed our system as an Internet resource accessible system.

Table 2.1: Comparison of the proposed work with previous works on infotainment systems.

Work	VANET	Cloud-based	Fusion	Data Adaption and Unification	Service Identification
Hackbarth [18]	No	No	No	No	No
Hsu et al. [21]	No	No	No	No	No
Sharma et al. [40] [41]	Yes	No	No	No	No
Festag et al. [13]	Yes	No	No	No	No
Macario et al. [31]	No	No	No	No	No
Lattanzi et al. [28]	Yes	No	No	No	No
Sharma et al. [42]	Yes	No	No	No	No
Sonnenberg [43]	No	No	No	No	No
Cheng et al. [9]	Yes	No	No	No	No
Huang et al. [22]	Yes	No	No	No	No
Salvo et al. [38] [36]	Yes	No	No	No	No
Amadeo et al. [4]	Yes	No	No	No	No
Kim et al. [25]	Yes	Yes	No	No	No
Ohn-Bar et al. [32]	No	No	No	No	No
Rangarajan et al. [34]	Yes	Yes	No	No	No
Al-Ani [2]	No	No	No	No	No
Hussain et al. [24] [23]	Yes	Yes	No	No	No
Proposed	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**

2.6 Summary

In this chapter, We gave a brief introduction about background knowledge related to this cloud based system. VANETs is our system environment, we make it clear that the Internet access support of network. We also stated some specific requirements for system of VANETs. Then, we introduced the system architecture we used to implement cloud based system. We brought the concept about mashup system which is used to extend the SOA. We discussed the current situation for infotainment systems. We mentioned that there is currently no unified platform for vehicle manufactures to develop infotainment applications. Also, most existing infotainment systems do not utilize Internet data. Furthermore, we also discussed two issues that limits manufactures who still use local data. Next, we talked about some related work and compared them to our work. For example, from the infotainment systems of Daimler Center and National Chia-Yi, we learned that we could improve our system by adding a middle ware server in whole architecture. From Gianpaolo Macario's work, we noticed that our work should be created using cloud based architecture. At last, we provided a comparison to show the differences between our proposed work and existing works.

Chapter 3

Proposed Architecture

3.1 Design Principles

The final goal of this thesis is to build an infotainment system that utilizes Internet resource through VANETs to enhance user experience. To achieve this goal, we need to face several challenges. In this section, we formulated the se challenges in the form of design principles. In order to state the design principles precisely, let us assume that $\mathcal{A}$ is set of applications, $\mathcal{R}$ is set of all Internet resources relevant to Infotainment applications, and $\mathcal{S}$ is set of services running in the cloud. In cloud-based systems, one part of overall application function is implemented at the server (using services) and the other is at the client. In this thesis, we name the client part as "Applications". With these definitions, we illustrate our design principles as follows:

1. **Data Fusion**: In order to provide accurate and up-to-date results, every service needs to fuse the information received from multiple Internet resources. If f_i is the function that implements fusion strategy, we can define each service as

$$S_i = f_i(\mathcal{R}_i) \tag{3.1}$$

   where $S_i \in \mathcal{S}$ is service and $\mathcal{R}_i \subseteq \mathcal{R}$ is set of resources required for service S_i. We want to find structure of f_i suitable for resources related to Infotainment applications.

2. **Context-Aware**: The client interface design and client functions should be context-aware so that the applications can bring the convenience for the passengers. If $\mathcal{C}_i$ is the contextual information required and F_i is the functionality implemented by application A_i, then, instead of having a separate application for each context, we intend to have context as a parameter of the application, i.e.,

$$A_i = F_i(\mathcal{C}_i, \mathcal{S}_i) \tag{3.2}$$

where $\mathcal{C}_i$ is set of contextual attributes affecting the application and $\mathcal{S}_i \in \mathcal{S}$ is the set of services used by application A_i. The context-aware filtering of the data should take place at the cloud to reduce data flow between RSU and OBU.

3. **Re-usability**: For saving the money and time in developing new application, our system should be designed for reuse, which means the services should be useful for multiple applications. The re-usability of a service is defined by the number of applications that use the service, i.e.,

$$\rho_i = \sum_{j=1}^{|\mathcal{A}|} |S_i \cap \mathcal{S}_j| \tag{3.3}$$

where ρ_i is the re-usability of service S_i. A high re-usability would also mean that the middleware is generic enough to support future infotainment applications.

4. **Loose Coupling**: Since the system can be deployed in various vehicles, we need to reduce the hardware dependence of the system so that the system is available to more vehicles. Furthermore, the system should be independent of Internet resources and their API's. This is achieved by having a lighter client size by delegating most computing tasks to the cloud. $T()$ is a function that returns the processing time of a functionality, the coupling degree κ_i is defined for i^{th} application as follows:

$$\kappa_i = \frac{T(F_i)}{\sum_{\forall j | S_j \in \mathcal{S}_i} T(f_j) + T(F_i)}. \tag{3.4}$$

We want to design services such that the infotainment applications developed using these services have low coupling degree.

3.2 System Design

In order to meet above design principles, we propose cloud-based framework for developing infotainment system. Since SOA is a software design pattern that reckons each application functionality as an independent service, in this design, one application is divided into a particular number of functional modules which individually represent a service. The application client delegates most of the services to the cloud which ensures a light weight client.

The block diagram of proposed framework is shown in figure 3.1. The proposed framework consists of three layers: (1) *Resource Layer* (set $\mathcal{R}$), (2) *Cloud Layer* (set services on this layer as $\mathcal{S}$), (3) and *Application Layer* (set $\mathcal{A}$). Resource layer represents physical sources from the Internet. Cloud layer implements services which consume resource

layer contents and finally produce suitable media for VANETs infotainment applications. Active service oriented interaction ensures up-to-date quality data. Here, the application layer runs on the client. Following are the important steps of the proposed framework:

1. A set of services are exposed in the main cloud.
2. A client application in the vehicle (i.e. OBU) invokes one or more services according to its functionality.
3. Service request is forwarded to the cloud and eventually retrieves data from multiple resources.
4. The retrieved data is filtered, fused, and processed at the cloud server.
5. Post-processed data is sent back to the application client.

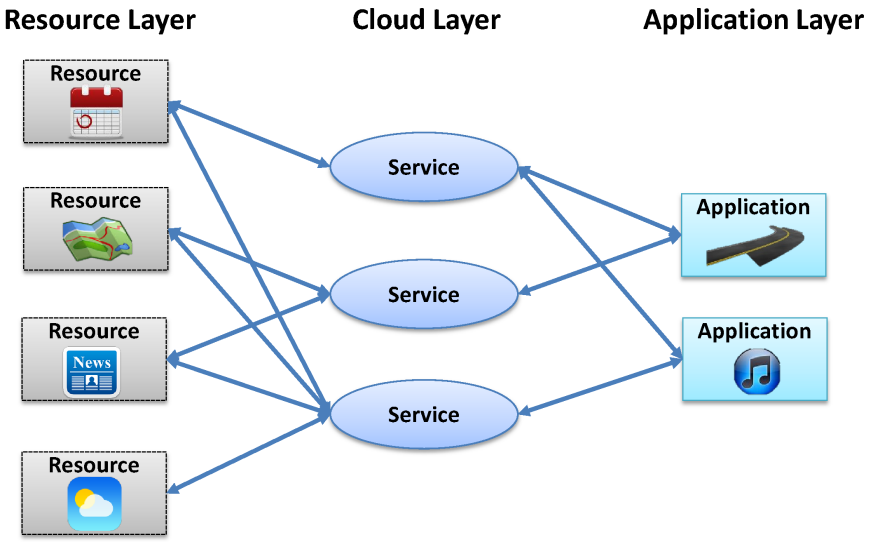


Figure 3.1: The block diagram of the proposed framework

In the step 3, data is retrieved from the Internet resources. Step 4 creates mashups of the retrieved information and publishes the final data through a composite service according to the requirements of vehicular infotainment applications. The cloud also performs some pre and post processing on the retrieved data to release loads off the VANET client. Note that only fine tuned data is transferred from the cloud to the client through VANETs communication channel. Hence, irrespective of the location of the cloud (RSU or Amazon), the data flowing over RSU-OBU link will be reduced.

In the following sections we describe each layer of the framework components incrementally with real implementation examples of application named *eDirection*. *eDirection*

provides latest up-to-date map routing service for City of Ottawa in Canada by fusing information from Google Map, MapQuest, Ontario Government Transport website and City of Ottawa website.

Example: Since navigation is the primary function service in current automobile, we show the implementation of *eDirection*, an application following InCloud architecture. *eDirection* service provides the user more reasonable routing result based on the information from multiple map information and routing data fusion. In the implementation, we take routing resources from Google Map and MapQuest. We also take the local road construction information from the Ontario Ministry of Transportation Website[1]. The outcome of this application provides better result after data fusion of multiple resources. We get the KML[2] file of the map and analyze the file to get the data about road condition evaluation and road constructions coordinates.

Table 3.1: Few examples of information resources $\mathcal{R}$.

Resources	**Description**
World Weather Online[3]	Weather information
Yahoo Weather[4]	Weather information
Google Directions[5]	Direction information and route calculation
Google Geocoding[6]	Address to location mapping information
Ontario ministry of transportation[7]	Road construction information
Google Calendar[8]	Event information
Youtube[9]	video, ratings, and reviews
Facebook[10]	community, pictures, and events
Twitter[11]	news, updates
GasBuddy[12]	Gas station information
lastfm[13]	Music information
CBC[14]	News

[1]http://www.mto.gov.on.ca/english/traveller/trip/map.shtml
[2]https://developers.google.com/kml/
[4]http://www.worldweatheronline.com/free-weather.aspx
[5]http://developer.yahoo.com/weather/
[6]https://developers.google.com/maps/documentation/directions/
[7]https://developers.google.com/maps/documentation/geocoding/
[8]http://www.mto.gov.on.ca/english/traveller/trip/map.shtml
[9]https://developers.google.com/google-apps/calendar/
[10]https://developers.google.com/youtube/getting_started
[11]https://developers.facebook.com/docs/reference/apis/
[12]http://www.gasbuddy.com/
[13]http://www.last.fm/api
[14]http://www.cbc.ca/rss/

3.2.1 Resources Layer

One of the design goals of the proposed framework is to reduce client side computation and increase the re-useability of the core implementation. That purpose is served by performing calculation in the cloud and providing data only request service to the OBU client. Resource layer refers to the *Internet* in figure 3.1. In this layer, we pick up the valuable resources. Web 2.0 makes data interaction convenient between *Internet* resources and users. Nowadays, more and more organizations offer their data to clients through REST, SOAP. Table 3.1 lists some of the popular information resources on the internet. Not only the clients, but also other servers can retrieve data through the web service protocols. But as stated before, resources can be chosen from different API providers considering that one is main provider and the others are complementary.

3.2.2 Cloud layer

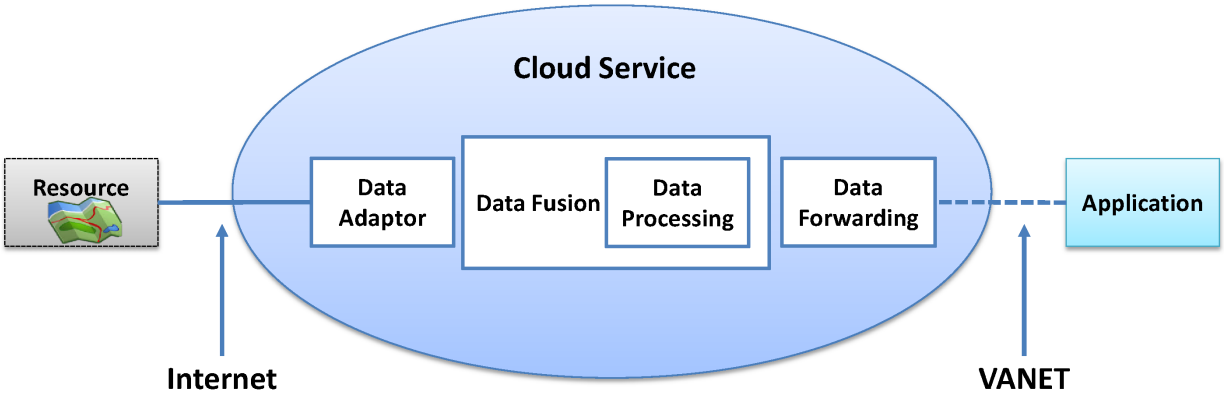


Figure 3.2: A functional view of a typical service in the cloud.

This is the core part of the proposed framework. The system removes the redundant data and fuse the information as the application requirement in this layer.We use mashup architecture to build new combined services for the vehicle client applications. Figure 3.2 illustrates internal components of the cloud layer. The services retrieve data from Internet resources and produce resulting data, which eventually flows over VANET communication link. To reduce the bandwidth requirements of infotainment content over VANET communication link and distribute processing, we apply mashup architecture in the operational cloud [50].

There are three important questions in the design of cloud middleware:

- What are the services provided by the cloud server?
- What are the main steps of a service?

- What functionalities does each step provide?

A service is more useful if it can be used by multiple applications. We have identified a list (Table 3.2) of appealing web functionalities which are useful in VANET infotainment applications. To implement these functionalities, we have designed 15 services as listed in Table 3.2. Though detailed design would be different, but each service shares three common steps: *Data Adaptor, Data Fusion*, and *Data Forwarding*. *Data Adaptor* is the component that interfaces Internet resources and *Data Forwarding* directs the generated final result to the client applications.

Data Adaptor

Data adaptor performs pre-filtering operations on the data received from the Internet resources. Normally, this block receives the data in JavaScript Object Notation (JSON) or XML format. In the pre-filtering process, we mainly perform two tasks:

- **Filtering:** Internet resources design their APIs for general usage. There is a lot of information which is not required in infotainment context. In this task we identify such information and drop it. Therefore, the data size is reduced, which is helpful to improve the data processing speed and data transmission speed. In the case of context-aware applications (i.e. time, location, or weather) irrelevant data are dropped. For example, we can filter the News updates based on current location (if user settings allowed) reducing the data flow over VANET link.

- **Unification:** After data adaptor, the information will be passed to the fusion block. Therefore, we convert information received from different Internet resources into a unified format for efficient fusion. The unified JSON file decouples Internet resources from fusion step. Algorithm 1 shows the detailed steps of the unification process in a cloud based service. In order to design any service we first identify the required parameters in that service (Select-Parameters). Then we expand each parameter to t more other words by using synonyms from *WordNet*[15] and store them (Expand-Parameters). Multiple languages can be integrated in this part as well. For every *WordNet* Synset or multi lingual words we identify a *main* word preferably in English (Expand-Parameters). Next Internet based resource parameters are extracted (Extract-Resource-Paramters) and we match the stored parameters with the retrieved parameters and copy their value to the unification data structure using *main* words.

Example The *getRoute* service of *eDirection* is hosted in the middleware server. Vehicle client adds parameters such as origin coordinator and destination coordinator in the request. Once given input, the application process starts. The server invokes Google map

[15]http://wordnet.princeton.edu/

]

Table 3.2: Important functionalities for infotainment applications on VANET.

Functionality	**Input**	**Output**	**Infotainment Applications**
Profile	User credentials	User preferences	Context-based Applications
News	Textual keywords	Latest news crawled from different news resources on the Internet	News Updates, Trip Planning, Events and Activities, Tourists Assistance
Calender	User credentials	Events compiled from different user calenders such as yahoo, google, facebook	Trip Planning, Alarms and Reminders
Geolocation	Textual address	Location coordinates	Trip Planning, Traffic Information, Tourist Assistance, Context-based Applications
Routing	Source and destination coordinates	A linked list of segments representing the optimal route from source to destination	Trip planning, Schedule, Time management
Ratings	Media file name	Ratings of the media file (music or video) from different review sites	Media playback, Media download, Media purchase, Playlist
Distance	Source and destination coordination	distance between source and destination	Trip planning, Time management
Weather	Location	weather information crawled from different resources	Trip Planning, Tourist Assistance, Context-based Applications

Algorithm 1 Data unification algorithm for k^{th} service

Require: $R \subseteq IR$ such that IR = Internet Resources
D = Internet Resource Data
Ensure:
1: $H_k \longleftarrow$ Select-Parameters(S_k);
2: $H_k \longleftarrow$ Expand-Parameters(H_k, t);
3: $I_k \longleftarrow$ Extract-Resource-Paramters(R_k);
4: **for all** $P \in H_k$ **do**
5: **for all** $Q \in I_k$ **do**
6: **if** $P == Q$ **then**
7: $main \longleftarrow$ find-main-word(Q);
8: append value of $D < Q, J >$ to $F < main, J >$;
9: /*where $J \in R_k$ */
10: **end if**
11: **end for**
12: **end for**
13: **return** Filtered Data, F

and MapQuest Map API, and gather direction results. It also collects the live data from MapQuest at the same time. Next, local road construction information is collected from Ontario Ministry of Transportation Website and City of Ottawa website. Figure 3.3 illustrates example data we get from Ontario Ministry of Transportation Website. Its format is KML file which belongs to XML file [17]. We try to get the KML file of the map and analysis the file to get the data like road condition evaluation and road constructions coordinates. We represent all the data in a common format which is part of unification and helps in the fusion phase.

Data Fusion

Data fusion step is vital for the proposed cloud-based middleware. The aim of fusion step is to combine different data sources and extract more comprehensive data for data processing step. This step further enhances the quality of data which might reduce the data size depending on the type of services. Table 3.3 shows the servers that we have implemented. For example, for ranking service the reduction ration will be $1/n$ for n resources. In case of route calculation we grab the road constructions data from Ontario Government Transport web site, while we also get the map data from Google. We fuse both information to obtain a route that is shortest and without obstructions. More details about the data fusion approach is covered in Section 3.3.

Example During the process of the data fusion, we take several pre-check steps to improve the system process inefficiency. We firstly make normal route search options

<?xml version="1.0" encoding="utf-8"?><kml xmlns="http://www.opengis.net/kml/2.2">
<Document xmlns="">
<name>Construction Information</name>
<description>Construction Information provided by the Ontario Ministry of Transportation</description>
<Style id="activeIcon"><IconStyle>
<scale>0.9</scale>
<Icon><href>http://www.mto.gov.on.ca/graphics/english/traveller/trip/icon-active_construction_contracts.png?5685685</href></Icon>
<hotSpot x="0.5" y="0.5" xunits="fraction" yunits="fraction" /></IconStyle></Style>
<Placemark>
<name>2011-2003</name>
<description>...</description>
<MultiGeometry>
<LineString><coordinates>
-79.610814,43.555320,131.28
-79.609329,43.557332,131.53
-79.554116,43.609825,138.77
</coordinates></LineString>
</MultiGeometry>
<styleUrl>#active</styleUrl>
</Placemark>
<Placemark>
</Placemark></Document></kml>

Figure 3.3: The example of the KML file.

Table 3.3: The implemented services and description

Category	**Services**	**Parameters**	**Resources**	**Comment**
News	getNewsByKey Word	keywords: value	Yahoo News, Google News, CBC News	Return the top news from yahoo news ,Google news and CBC news based on the input keywords
Music	getRMusic	artist: value, name: value	Musicovery, last fm	Return the recommended music based on the users preference and most played music style
Music	getPMusic	song: value	xiami	Return the online media link. The song contains the songs name and artist name
Music	getPlayHistory	userID : value	Local Database	Return the user play list history
Route	getRouting	originLat: value, origin-Lon:value, destination-Lat : value, detination-Lon:value	Google Map, MapQuest, Ontario ministry of trans-portation	Return route result from Google Map and MapQuest, optimized by road construction information from Ontario ministry of transportation
Route	geolocation	place:value	Google Map	Return the formatted address using Google Geolocaiton service. The place value can be coordinators or place political name.
Route	getNaviSettings	null	Local Database	Return the setting for the navigation
Route	setNaviSettings	null	Local Database	Update the setting for the navigation

Table 3.4: The implemented services and description

Category	**Services**	**Parameters**	**Resources**	**Comment**
Weather	getWeatherInfo	Lat:value, Lon:value	Forecast, world-weatheron-line, wunder-ground	Return the weather forecast result, pick up the resources from Forecast, worldweatheronline, wunderground resources based on the data trusted calculation
Weather	getYahooWeatherID	latlon:value	Yahoo, Google Map	Return the Yahoo Weather WOEID
Account	loginCheck	username: value, password: value	Local Database	Return account check result and user ID if account is valid
Account	getAccountInfo	userID:value	Local Database	Return the users information based on the user ID
Calendar	getEventInfo	userID:value	Google Calendar, Local Database	Return Google Calendar event items information
Calendar	getEventPlace	userID:value	Google Calendar, Local Database, Google Map	Return Google Calendar event place formatted address

based on the users preferences. Then we do an precise check if there is any construction ongoing in the current route bounds. If there is a construction ongoing, we change the routes to see if we can avoid the road construction.

Data Forwarding

In the VANET infotainment application, the application request comes from the vehicle client which is registered in the cloud. The vehicle client makes the request to the current RSU along with its position, speed information. In our proposed framework we support two types of cloud hosting. The straight forward approach is to have a central cloud for data processing and result forwarding. Another approach is to host the clouds in the RSU in a more distributed manner. Detail description follows:

- In the former approach, RSU forwards the request to the cloud server which identifies the dynamic geolocation of the the requesting vehicle client and its approaching direction in the connected RSU. Using this information, the cloud server determines the possible next handover RSU in the traveling path of the vehicle. Next, the cloud server replicates the information in both current RSU and next RSU along with reusable cached information for independent RSU.
- In the later approach, every RSU works as an infotainment cloud. When a request comes to this RSU cloud, it determines the direction of the vehicle and forwards the processed data to the requesting vehicle. The processing RSU cloud also replicates same information along with intelligent approximation and directs it to the probably next RSU.

Example The route data is transferred to the RSU middleware server and from there sent back to the vehicle client. In our implementation we only considered one RSU. In multiple RSU scenario, we will have advanced forwarding which is part of our future work.

3.2.3 Application layer

In *InCloud* we intend to make application clients as lightweight as possible to achieve loose coupling with the underlying hardware of the vehicle. Application clients mainly implement modules related to user interface and information presentation. The client may also include processing steps such as video/image decoding which is not done at the server because of large size of the decoded data. Recent attention[16] of consumer electronics giants Apple, Google, and Microsoft in building operating system for car dashboard, which makes

[16]http://www.slashgear.com/google-projected-mode-android-tipped-to-take-on-apple-in-cars-02319097/

```
$.ajax ({
    type:     "POST",
    url:       http://"+serverIP+"/infotainment.asmx/+servicesName+?jsoncallback=?",
    data:      parameter,
    dataType: "jsonp",
    jsonp:    "jsoncallback",
    success:  function (data) {},
    Error:    function (err) {}
});
```

Figure 3.4: The example of Ajax call pattern

our system more relevant since multiple platforms can consume our services to build their infotainment applications. Our light weight client approach ensures interoperability among multiple platforms as well.

The infotainment client is implemented on android platform. We host the client on Galaxy Tab 3 mobile device. The mobile device also has two network interfaces: 3G and wireless. However, at any given time only one interface is active. The client maintains a WiFi Direct connection with the RSU server. The client implements the user interface and data processing, depending upon the application. The applications use AJAX with XMLHttpRequest protocol to interact with the IIS web server in RSU which is common for all web services. With the help of web browser and these technologies, users can easily address rapid data interaction. We have chosen to use AJAX because it enables us to update a part of the webpage without refreshing the whole page. Hence, the reaction speed is much faster than before.

In the server, we provide the web service interface by both REST and SOAP. For the services' result, we use JSONP. It is JSON with padding. The difference is that we add a header or a pair of parenthesis around the JSON string to generate JSONP string. JSONP allows the client to load the JSON feedback as a script file. It is usually used for cross-site AJAX with JSON data. In the implementation, our client use ajax from jquery library. JSONP makes it easier to invoke the web services. Based on the services tables, our web sevice URL pattern: "Http://+ **ServerIP** + / infotainment.asmx/+ **servicesName** +?jsoncallback=?". In the client, the ajax call pattern is shown in figure 3.4

3.3 Data Fusion

Data fusion strategy depends on the type of information being fused. We can broadly categorize the information resources into three types: resources that provide opinions, resources that provide auxiliary information with respect to each other, and resources that compliment each other with additional information. Description of these resources and examples are given in the Table 3.5.

Table 3.5: Resource types from fusion perspective.

Resource Type	**Description**	**Examples**
Opinions	These resources provide opinion about an event, activity, or object.	Music ratings, movie ratings
Auxiliary Information	These resources just add more quantity.	News, point of interest
Complimentary Information	These resources compliment each other towards a common goal.	road map, traffic info, road construction

Each service may need to use different fusion strategy depending on the type of the information provided by the resources. It is also possible for one service to receive data from multiple types of resources. Therefore, a service may have all three types of fusion strategies. The fusion model components are categorized as:

1. Automatic reputation models [45] or prior knowledge are given as *Input* to choose the main resource.

2. *Classify* the remaining resources according to the type of information provided as given in Table 3.5.

3. *Data Processing*: first fuse opinions, then auxiliary information, and finally complimentary information.

Figure 3.5 shows overview of the fusion method employed by the services and Algorithm 2 shows the higher level algorithmic steps. First we identify the main resources recommended for a service using primary knowledge and reputation model (Find-Main-Resource), next we identify opinion (Find-Opinion-Resource), auxiliary (Find-Aux-Resource), and complimentary services (Find-Comp-Resource). Later, based on the application type requirements, we identify the composition of opinion, auxiliary and complimentary information (Find-Mode). These additional information are later fused with the main extracted information in a specific order which is consumed by the vehicular client (Φ_o, Φ_a, and Φ_c). Every vehicle client is assigned an unique key which helps to log earlier interactions in the internet cloud. These early records help in various fusion operations. Important components of the fusion methods are described below.

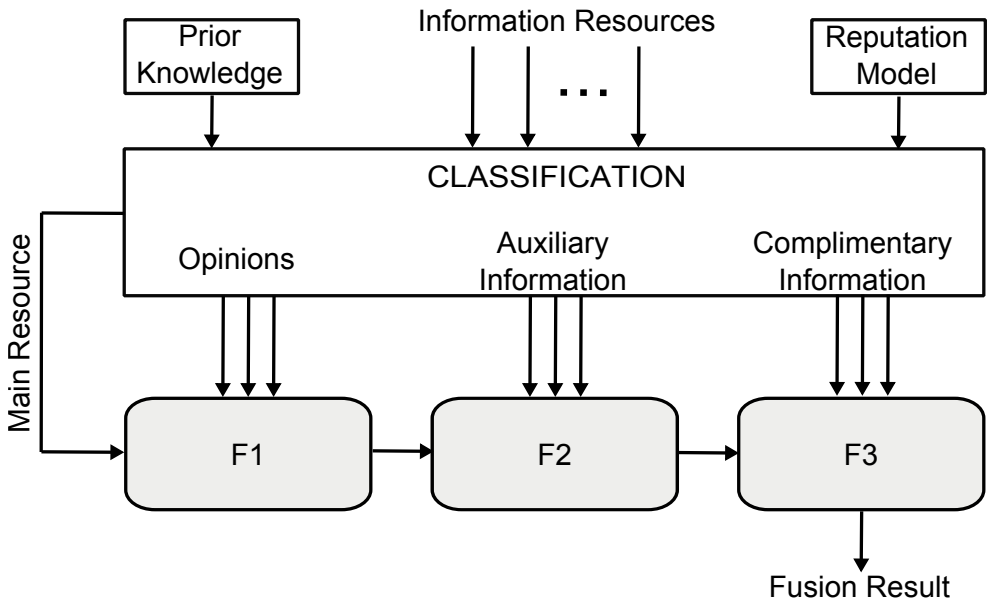


Figure 3.5: The fusion takes place in three steps depending on the type of resources.

Algorithm 2 Data fusion algorithm for for k^{th} service

Require: $R \subseteq IR$ such that IR = Internet Resources PK = Primary Knowledge
RM = Reputation Model

Ensure:

1: $R^m \longleftarrow$ Find-Main-Resource(R_k, PM, RM);
2: $R^o \longleftarrow$ Find-Opinion-Resource$(R_k - R^m, PM, RM)$;
3: $R^a \longleftarrow$ Find-Aux-Resource$(R_k - R^o - R^m, PM, RM)$;
4: $R^c \longleftarrow$ Find-Comp-Resource$(R_k - R^o - R^m - R^a, PM, RM)$;
5: $Mode[o, a, c] \longleftarrow$ Find-Mode $(ApplicationType)$;
6: $F = Extract(R^m)$;
7: **if** $Mode[o] == True$ **then**
8: $\quad F = \Phi_o(F, R^o, PK)$;
9: **end if**
10: **if** $Mode[a] == True$ **then**
11: $\quad F = \Phi_a(F, R^a, PK)$;
12: **end if**
13: **if** $Mode[c] == True$ **then**
14: $\quad F = \Phi_o(F, R^o, PK)$;
15: **end if**
16: **return** Fused Data, F

3.3.1 Fusion Input

We integrate various information in the data fusion, following is the detail of the input information for data fusion.

Prior Knowledge

In our system, users incorporate their prior knowledge about the quality of the data provided by different web resources. This information is stored in the Internet cloud against the vehicle client key which can be reused or updated over time. For instance, we can experience that one news channel provides information of more interests than others. Similarly, during our earlier trips we might have experience that a particular website suggests better places to visit than other websites. Prior knowledge is retrieved from the Internet cloud whenever necessary. It is also required to determine the type of any resource.

Reputation Model

The system also learns the reputation of each web resource by observing user returns over time. For instance, the cloud may compare the places visited by a person and the places suggested by websites over time. There are also scenarios where the system may determine the correctness of partial information provided by the web resource and build reputation. For example, the vehicles generally have sensor to measure outside temperature accurately. This temperature value can be compared with the one provided by the weather resource over time to identify which resource provides more accurate information. This can help in identifying more trusted information resources.

3.3.2 Classification

This component classifies the resources according to the prior knowledge and reputation models. It selects one resource as the most trusted main resource and classifies the rest with respect to the main one. It enables and disables each fusion step according to the availability of the resources. For example, opinion fusion step is only enabled when there is a resource providing opinion information, otherwise the main resource data is forwarded to the next step. Same strategy is employed for each fusion step.

3.3.3 Data Processing

Opinions

This is the first step of the three steps fusion method. If there is any additional resource that provides opinion about the event, activity, or object under consideration, we need to

fuse that information to derive a single conclusion. For fusing opinions, we propose to use weighted some of individual opinions. The weights can be calculated based on the website reputation model or prior trust value assigned based on past experience. If there is no resource providing opinion information, this step is disabled and the input data is copied to the output data.

Auxiliary Information

As the web technology is growing rapidly, more content is now available online. For example, a large number of news websites report current events and happenings. As a result, many web resources accumulate to greater number of information which can provide the user with more options. If auxiliary information is preferred by user settings then we can get more personalized content by applying user profile and preferences on the data. For instance, choosing news based on current location and interests of the user.

Complimentary Information

Different resources can provide different data/media which in turn completes the main trusted information. For example, one resource provides road map, another traffic restrictions, and third one provides ongoing road construction. All three types of information fused together provides the best up-to-date route information. For every category of functionality, we need a separate fusion strategy. In the implemented system *eDirection*, *getRoute* service receives map and traffic data from Google Maps and road construction information from Ministry of Transportation respectively.

3.4 Summary

We have described a cloud-based infotainment system in this chapter. The middleware is developed according to the system design principles discussed earlier, such as cross-platform feature, system compatibility, and service re-usability. This framework is created using SOA extended with mashup methods. We divided this framework into three layers and listed all the related resources or methods in each layer. Second, we provided details of the fusion method used. Finally, we illustrated our prototype system. We showed the connection method between resources layer and cloud layer and listed main services we created in the implementation.

Chapter 4

System Implementation

In our implementation we consider RSU as the middleware server which is a laptop with Windows 8 OS, Intel Core i7-3820QM processor, and 4 GB RAM. The laptop has two network interfaces and simulates RSU in VANETs [37]. The wired Interface provides Internet connectivity while the wireless network is used by clients. Our web services are built using Microsoft Web Platform and hosted in Internet Information Services (IIS) 8.5. The services are written in C# object oriented programming language using Visual Studio IDE.

The client can connect to the server and request any service through AJAX. The client also sends data with the request depending on the request. For example, to request functionality of *getDistance* web service, the client supplies source and destination location to the server. After evaluating, the web server sends results back to the client through JSON file.

To create service mashups, we write our own mashup code except news service. For news service we use Yahoo pipes. For all other services, we first fetch data from different resources through API's, RSS feeds, or by grabbing web data. The list of all services and their implementation details are provided in table 3.3. The source code of the server is available on Git public repository[1] on www.bitbucket.org for download and testing. Here we give the class diagram to illustrate the organization of the service as in figure 4.1.

We have implemented fifteen different services (Table 3.2) in different application domains among which *eDirection* is described inline with the framework and two other applications (i.e. music player, and news service) are described in detail to validate the efficacy of the proposed infotainment middleware. Figure 4.2 represents the architecture of our implemented VANET infotainment system. Here *Vehicle Client* is a tablet and information resources come from *Internet*. Any request goes from client to the *RSU*. Both *RSU* and *Server* can play the *Middleware Server* role. In our testbed, we have considered RSU only and hosted the middleware server in RSU. All the resources are consumed from Internet.

[1]https://bitbucket.org/mksaini/incloud-infotainment-middleware

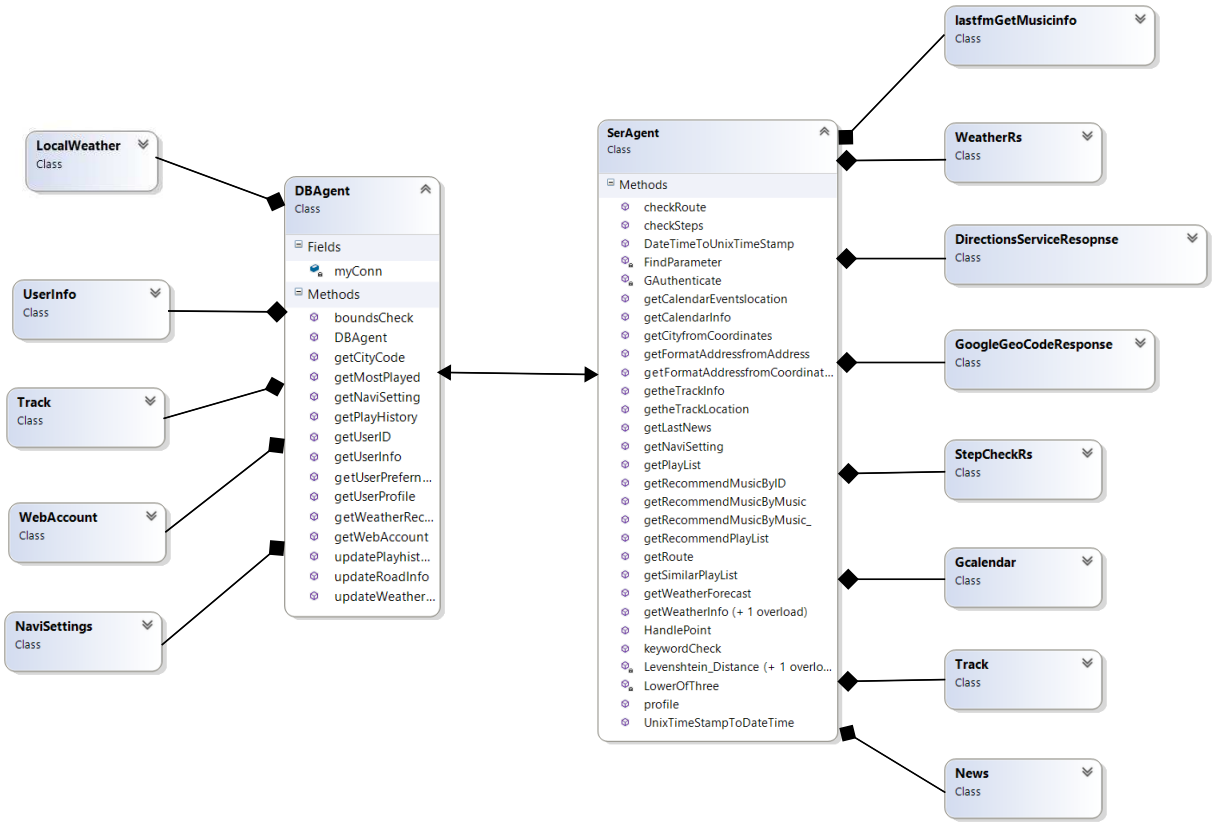


Figure 4.1: Class diagram of infotainment system.

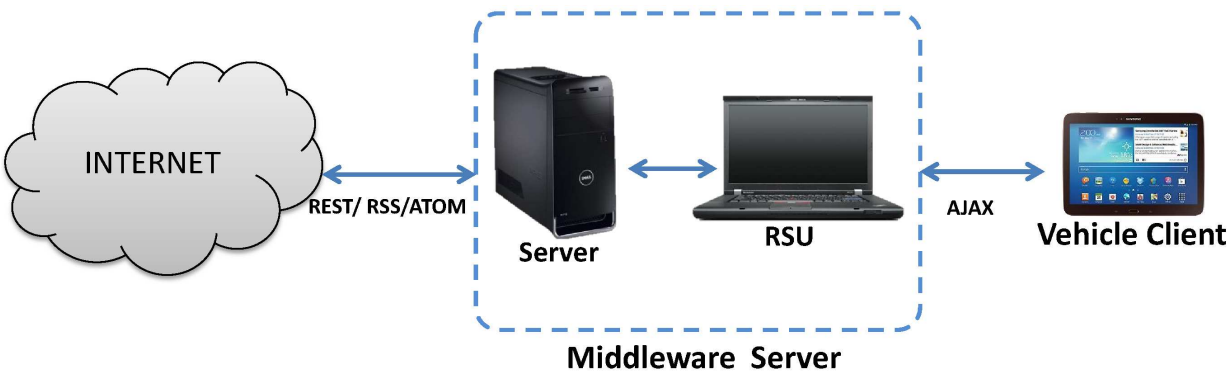


Figure 4.2: Implementation of infotainment system.

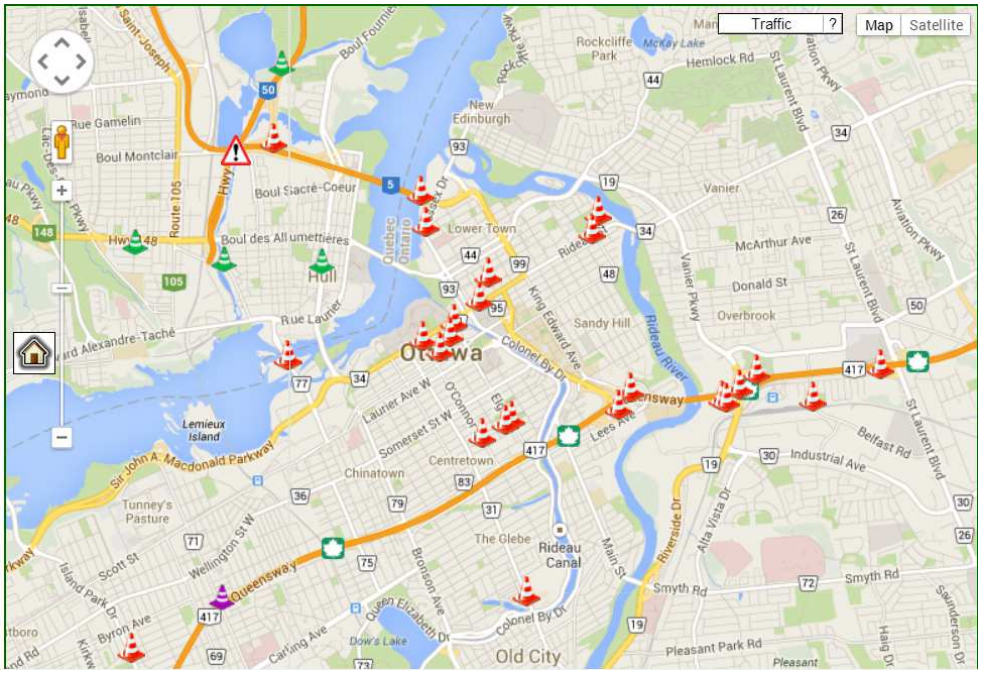


Figure 4.3: Screenshot of the map website from City of Ottawa.

4.1 Context-Aware Routing Application

The routing application is to provide users a new route which can avoid the additional constructions data. Fusion Input is user preferences such as faster service, toll free service etc. The system selects the main resource following this prior knowledge. Classification uses the fusion input data and identify Google Map as main resource, MapQuest as auxiliary information, and City of Ottawa as complimentary information. As shown in figure 4.3, we can observe that there are construction information added to Google Map. The whole flow of the application is shown in figure 4.4:

a) At first, server retrieves the route bounds which is a pair of coordinates. The two points will generate a rectangle while they work as south-west and north-east corners. We can see from figure 4.5. The black rectangle is the route bounds. The server will calculate whether the construction points fall in this area or not. Let north-east point be (X_{NE}, Y_{NE}), south-west point (X_{SW}, Y_{SW}) and construction point (X_C, Y_C). Then we get the distance of two points like (X_{NE}, Y_{NE}) to (X_{SW}, Y_{SW}) is D_1 , (X_{NE}, Y_{NE}) to (X_C, Y_C) is D_2, (X_C, Y_C) to (X_{SW}, Y_{SW}) is D_3. So if $D_2 < D_1$ and $D_3 < D_1$, the construction point falls in this area.

b) If there is no construction points in this area, system will send back the result. Otherwise, the system will divide the route into parts based on the turning and distance. Each part called step has its own start point coordinators and end point coordinators like the red rectangle in figure 4.5. As shown in figure 4.5, the area covered by red rectangles is

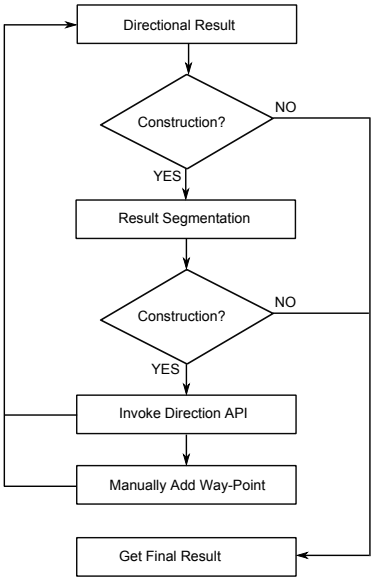


Figure 4.4: Flow chart of eDirection proposed algorithm.

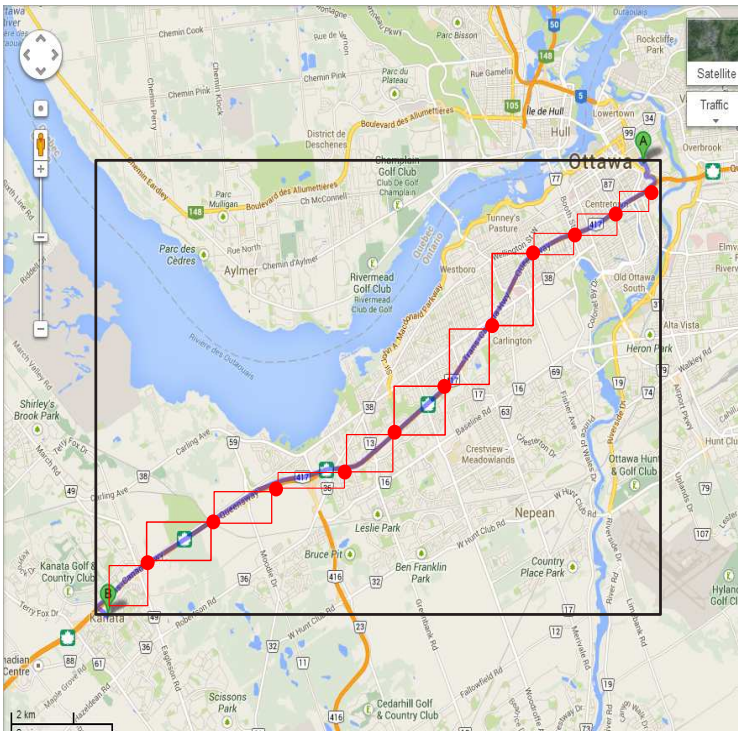


Figure 4.5: Example of step area.

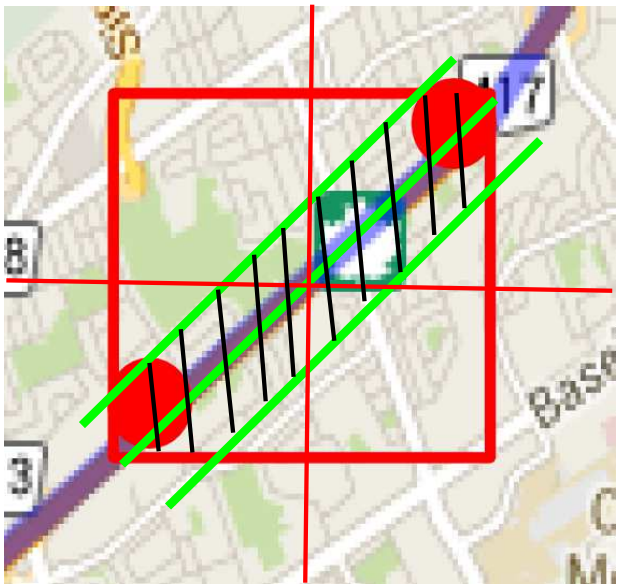


Figure 4.6: Detail of the step calculation.

obviously smaller than black rectangle. We can reduce the construction check area in this way to make more accurate construction check. Theoretically, if we red rectangles number approaches infinity, then the red rectangle area approaches the route.

c) Next, system will try to search which step locates the construction. Again, if there is no construction points in these steps, system will send back the result. Once the construction locates the step, system will do further verification. The bending angle of the route in each step is less than 90° since the step is divided based on turning. We assume that the route in this block is near the diagonal of the step. So next we want to verify if the point falls in the shadow area like in figure 4.6. For each one construction point, we set a radial with it as starting point which is paralleling X axis. For each side of blue polygon, we calculate if there is intersection with the radial. If the sum of the intersections is odd, the it means the construction is in the polygon. If it is even, then the construction locates out of the polygon. Algorithm 3 shows checking constructions points in each red area:

d) Last, we try to make new route to avoid the construction point. First, we generate new route using MapQuest API, redo the upper steps to verify the route. Second, if new route can't get through the verification, we create route by adding new way point. The figure 4.7 shows that we assume construction is in red area, we add a new way point out of this area. We first add one new point in one side of the area, test if the new route can avoid the construction point, if not, we clear old way point and add new one in the other side. Then again do the verification. If still does not work, we repeat upper steps until the new route can pass the verification. When all the step segments are clear, the system send back the final route result. Figure 4.8 displays the process of system. Following is pseudu-code for route optimization:

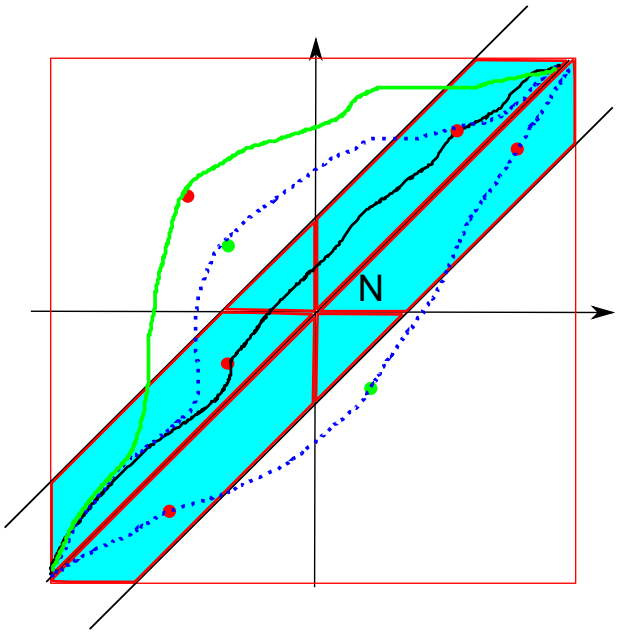


Figure 4.7: Detail of the step calculation 2.

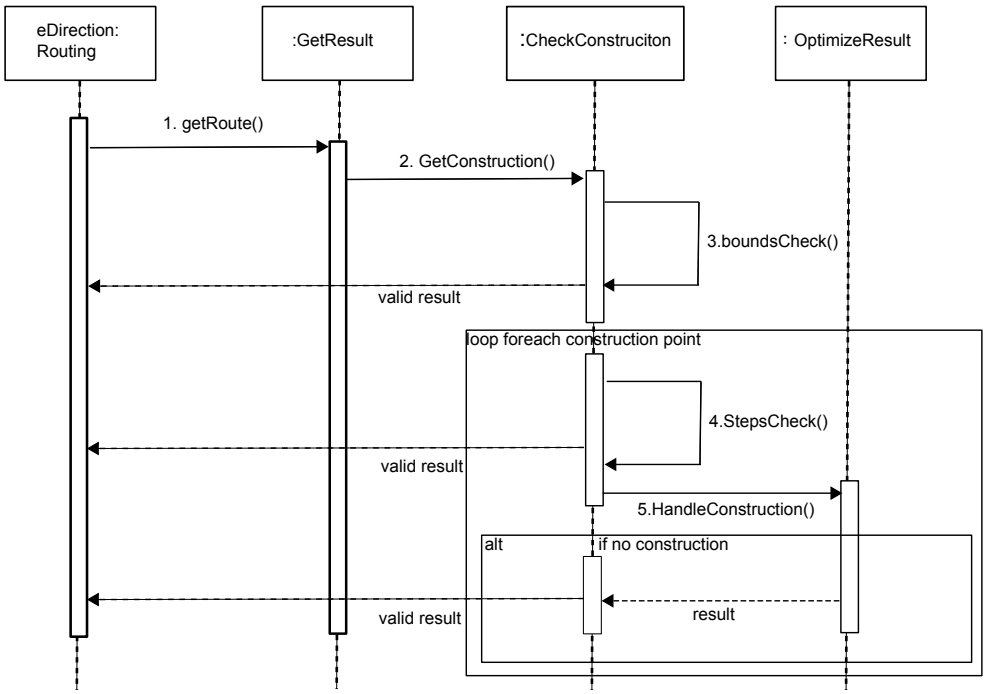


Figure 4.8: Interaction diagram of eDirection.

Algorithm 3 Construction Point Check

Require: list of construction points' coordinators ω_c $\omega_c = \{(S_1, S_2, \ldots, S_k\}$, $S_k = (X_k, Y_k)$ area start point coordinators (X_s, Y_s) area end point coordinators (X_e, Y_e) new way point coordinators (X_N, Y_N) sides set of polygon $\beta = \{L_1, L_2, \ldots, L_n\}$ set tolerance t.

Ensure:

SET J=0;
for all $S_k \in \omega_c$ **do**
  SET RADIAL $Y = X_k$;
  for all $L_n \in \beta$ **do**
    if $X_N < 0$ AND $Y_N > 0$ **then**
      $K_L = |Y_N - Y_s|/|X_N - X_s|$ or $K_L = |Y_N - Y_e|/|X_N - X_e|$;
      $L_n \rightarrow Y = K_L X + (Y_N + 2^(1/2)t) - K_L(X_N - 2^(1/2)t)$;
      CALCULATE IF RADIAL $Y = X_k$ has intersection with L_n;
      intersection EXISTED
      J= J+1;
    end if
    if $X_N > 0 AND Y_N < 0$ **then**
      $K_L = |Y_N - Y_s|/|X_N - X_s|$ or $K_L = |Y_N - Y_e|/|X_N - X_e|$;
      $L_n \rightarrow Y = K_L X + (Y_N - 2^(1/2)t) - K_L(X_N + 2^(1/2)t)$;
      CALCULATE IF RADIAL $Y = X_k$ has intersection with L_n;
      intersection EXISTED;
      J=J+1;
    end if
    RS = J MOD 2;
    if RS !=0 **then**
      $\omega_c = \omega_c - L_n$;
    end if
  end for
end for

Algorithm 4 Route Optimization

Require: area start point coordinators (X_s, Y_s) area end point coordinators (X_e, Y_e) set construction check result $\beta = \{S_1, S_2, \ldots, S_k\}$ set $\alpha = |Y_s - Y_e|/|X_s - X_e|$ set tolerance t

Ensure:

SET J=0;
GET THE MIDDLE POINT COORDINATORS $S_m = ((X_s + X_e)/2, (Y_s + Y_e)/2)$;
for all $S_k \in \beta$ **do**
  while $\beta \notin \phi$ **do**
    J=J+1;
    $S_m = ((X_s + X_e)/2 + J * t, (Y_s + Y_e)/2 + J * t)$;
    FOR $((X_S, Y_S), (X_m, Y_m))$ AND $((X_m, Y_m), (X_e, Y_e))$;
    CALL CONSTRUCTION METHOD 3;
    if $\beta \in \phi$ **then**
      $\beta = \beta - S_k$; Break;
    else
      $S_m = ((X_s + X_e)/2 - J * t, (Y_s + Y_e)/2 - J * t)$;
      FOR $((X_S, Y_S), (X_m, Y_m))$ AND $((X_m, Y_m), (X_e, Y_e))$;
      CALL CONSTRUCTION METHOD 3;
      if $\beta \in \phi$ **then**
        $\beta = \beta - S_k$; Break;
      end if
    end if
    Continue;
  end while
end for

With respect to cloud technology, infotainment system can gather massive data so that it can offer user more accurate and high-quality context aware services. Besides the routing service, we also implemented other services as shown in the services tables. Following figure 4.9 is the use case diagram of our system.

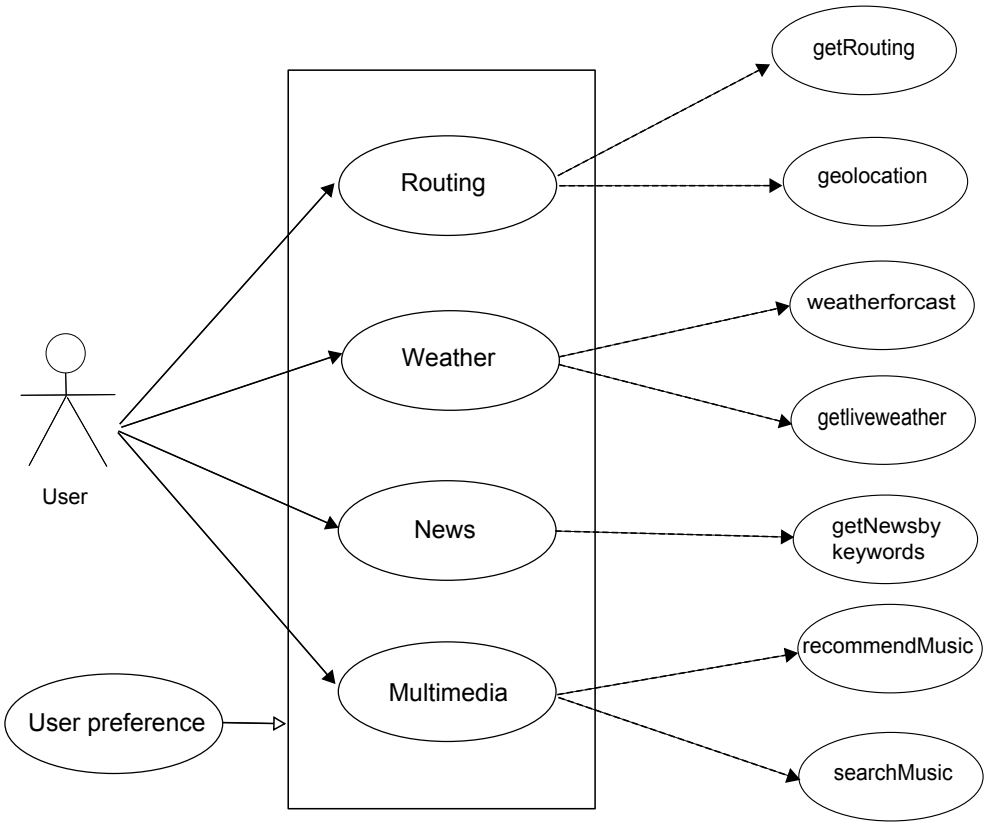


Figure 4.9: Use case diagram of system

4.2 Context-Aware News Application

News service can provide the recent news based on users' preferences and location. Suppose that a driver turns on the system, it will fetch the user's information from server. According to user's operation history, system will generate the categories of News that user viewed most. Also the system will catch the latest news based on user's current location. For verification of this service, we created a function which searches the news by keywords. We use Yahoo Pipe, an online mashup tool, to maintain the data fusion task.

With the development of the mashup system. More and more IT companies such as Yahoo, Google, IBM, start to create their own mashup tool. Yahoo pipe is one of them. We choose this online mashup tool for the following reasons: 1) News service needs little

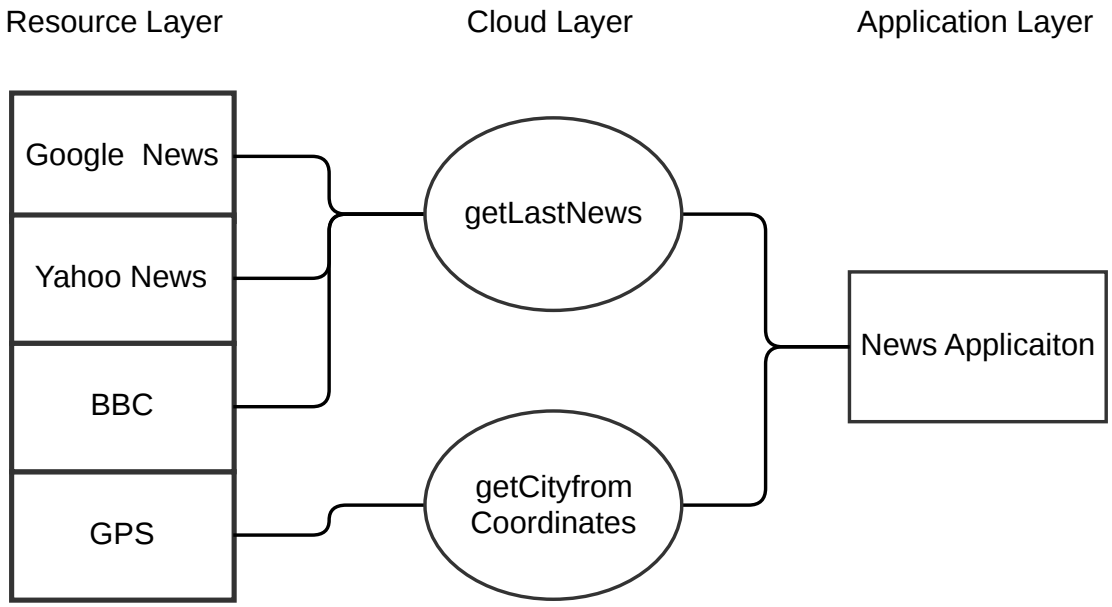


Figure 4.10: The schematic diagram for news service

data processing which means we do not need much calculation on the news data. This service only needs to fuse and filter the news content. Yahoo pipe can improve the system efficiency. 2) Using Yahoo pipe gives us convenience to access Yahoo search engine. Figure 4.10 is schematic diagram for news services. It illustrates how the system fuses multiple resources using different services. The process is as follows:

1. In the first step, the client sends the request to the mashup server. The input parameter represents the keyword. In the server, if no parameter is present, the system will take the top news as default value. The news service will firstly call "getCityfromCoordinates" method. This method invokes the Google Geocode API that can get the users' current location coordinate.

2. Next, the service will call the method "getLastNews" with the list of keywords. If the keyword list is empty, we will look up in the users' history to find highest frequency keyword for news. Then recall the "getLastNews" method. For the "getLastNews" method, we gather the top news from three news search engine: Google News, Yahoo News, BBC News. For the free resources purposes, we use different approaches to fetch the data. For instance, we use API for Google, RSS and Feed files for BBC and Yahoo News. For this step, resources firstly are gathered in Yahoo pipe. And the server gets the RSS file from Yahoo pipe.

3. After gathering the data, we unify the data. We set news object that includes : Title, Content, Link, Publisher, Publish Date five parameters. No matter what source of news we get, we just pick up these five related information elements and initialize

the object. We can get latest 50 viewed news title and subtitle and analyze the new category and keyword.

4. Last, the system will check the duplicated news based on Link, Title, Publisher. After getting the result, it check the news link and remove duplicated ones. Then server send the result back to client. In client side, we try to minimize the operation of the passengers.

We can see the news services are context aware since the news search result not only relies on the keyword but also relies on the users' GPS data and preferences. The news services can provide users more useful news according to his location.

4.3 Context-Aware Music Application

Since the vehicles can access to the Internet, plenty of multimedia data will be available through the Internet. Besides the music search service for expected music, it is very important to have a music recommendation system based on users' preferences. Considering the safety and convenience, it is better for passengers to have one-click application instead of trying the search content on the screen. However, most music recommendation systems give the music based on keyword search history or music type. Some audio database can provide the music rating service. Yet, the limitations of the region, languages, ages of the users, still hinds online music from providing convincing result. For instance, `www.xiami.com` is Chinese on-line music website which is similar to the Lastfm. "JuanZhuLian" is very popular song and has very high rank in this website. While we can hardly find the information from lastfm. On the other hand, the people who like 60s' music have hardly used online website like lastfm. Since there is not many play records of these tracks, the music they like mostly does not have good rank. Due to these kinds of reasons, single on-line music resource may not provide users a satisfied recommendation result.

Figure 4.11: The multimedia service flow chart

Multimedia is one important component of the infotainment system. To enhance QoS, we try to provide users with better music recommendation service. We design a music recommendation service to provide users new music that fits their desires. The on-line music service fetches the onine music by taking the advantages from the mashup system. Combining the user's play history, recommendation service can provide passengers high rate musics which are similar to users' favorite style. Figure 4.11 illustrates the flow of whole process. As depicted in figure 4.12, the music service process as following:

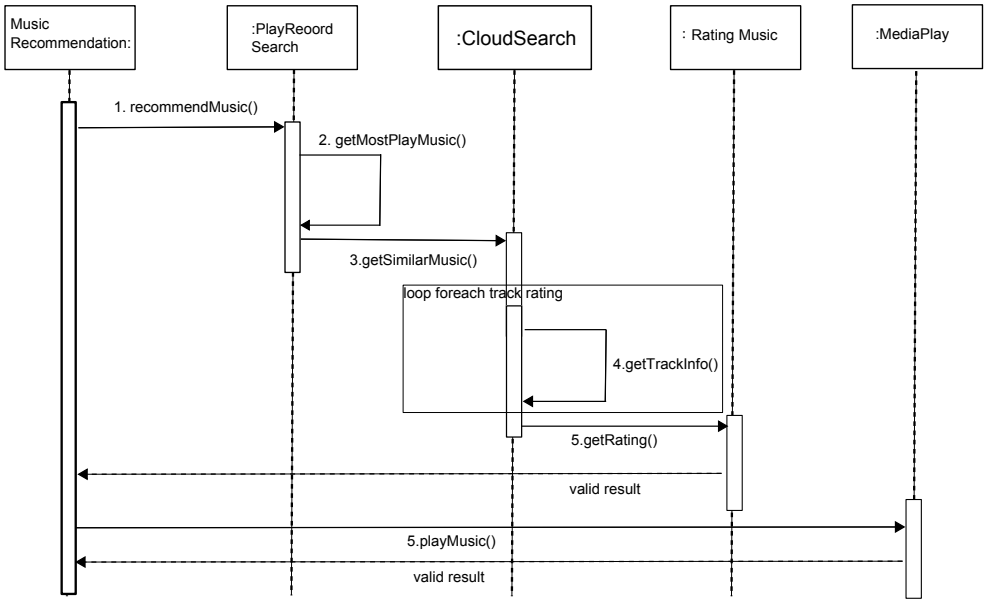


Figure 4.12: The interact diagram of recommendation music service

1. Once the server receives the music recommendation request from the vehicle client, it searches user's playback history from the database and finds the most played song.

2. The server searches for similar songs on Musicovery[2], a music search engine. Then the server gathers each song's popularity from Musicovery. The popularity value ranges from 0 to 100, where 0 means new or unpopular song and 100 means a hit song. We denote the popularity value from Musicovery as S_n^1, where n is the song index number.

3. We also get the play counts from the Lastfm. Since the play count is incremental, we use a function to limit the scope of the value between 0-100. If maximum play count is denoted as S_m^2 and current play count as S_c^2, then the popularity value from Lastfm is calculated as: $S_n^2 = (S_c^2/S_m^2) \times 100$. The final popularity rating R_n is calculated as:

$$R_n = \sum_{j=1}^{r} w^j S_n^j. \tag{4.1}$$

   where r is number of resources and $w^1 + w^2 .. + w^r = 1$ are weights assigned based on prior knowledge. In our application, we take two resources and equal weights, so our rating is $R_n = (S_n^1 + S_n^2)/2$.

4. After calculating the final rating, the server sorts the songs based on the ratings and returns the first ten results to the vehicle client. Each item contains the basic track

[2]www.musicovery.com

information, such as track name, artist name, album image etc. Once the user clicks any item on the playlist, the server requests for the online music service. This service also searches for online music streaming resource and gives the resource link back to the client. In this implementation, the service finds the resource form xiami.com.

Due to the multimedia copyright issues and free resources issues, our system only can find limited Internet resources. But this is enough to verify the concept of proposed service. If the system is established in commercial use or supported by more media web resources, the server can get more feedback and it will give more comprehensive music rate result. Also the on-line media search will have a larger scope.

4.4 Context-Aware Weather Application

For the weather service, accuracy of results is the key to QoE. More accurate data can also influence the data fusion result that uses the weather data as parameters. We build a trusted system to choose more accurate weather resource to use. Our trusted system model is to record the weather history data from different resources. It compares them with vehicle sensors data, pick up the one resource most close to real outside vehicle data.

One important element of context aware application is location awareness which means the location information takes part in the data process of the application. Context-aware also means synthesis of different service information. This can simplify the users' operations and improve users' efficiency. We will present a scenario to give further explanation. There are two methods in this service: getWeatherInfo and getYahooWeatherID. getWeatherInfo is a meant to return the user trusted weather forecast data. The figure 4.13 and figure 4.14 illustrate the flow of the getWeatherInfo service:

1. The service will firstly invoke the geolocaiton method from the Route service using the vehicle's GPS. The geolocatioin webservice from Route service returns the formatted address of the user's current location.

2. Next, the server will search the local database for temperature record of the last 5 days. This recorded data comes vehicle sensor. Every hour the server will receive the record temperature data from the vehicle sensor, if this value is higher than the maximum value or lower than the minimum value in the database, the server will update the record. Therefore, the database can store maximum and minimum temperature. We define these two value as $MaxTemp_n$ and $MinTemp_n$, n means the day from history, usually $n = 1, 2, 3, 4, 5$ which means we take 5 days history record from the database.

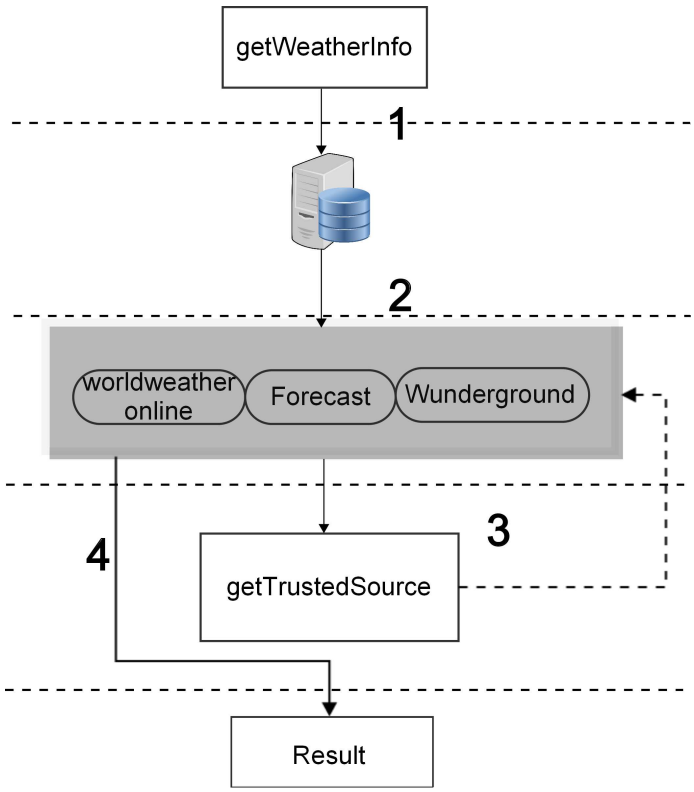


Figure 4.13: The flowchart of the weather service

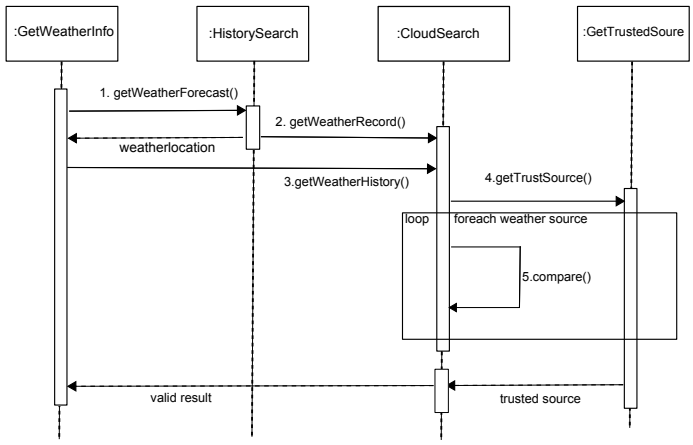


Figure 4.14: The interact diagram of the weather service.

3. After second step, we need to fetch the online resources history result. In the proof of concept, we consider weather information from 3 resources: Forecast[3], Worldweatheronline[4], and Wunderground[5]. If we set the maximum temperature and minimum temperature as $MaxV(K)_n$ and $MinV(K)_n$, K means the number of the weather resources, n means the day of the history. For example the maximum temperature of yesterday, the first day back from today, from Forecast is $MaxV(1)_1$, and the criterion value is $V(k)$. Then we have $V(K) = \sum_{n=1}^{5}(|MaxV(K)_n - MaxTemp_n| + |MinV(K)_n - MinTemp_n|)$.

4. After our calculation, we choose the minimum $V(K)$ value as our trusted resource. And we then get weather forecast of user's current location using the sources from the trusted resources. Figure 4.15 shows one week weather forecast result.

GetYahooWeatherID service aims to give the client an ID which can active the yahoo weather widget. For the current weather information, there are already many mature weather widgets that provide users instant weather information. Since this data needs to be redirected from the Internet to the client by server, we just return the API key instead of whole package of the data to client to request the weather data. This may reduce the consumption of the bandwidth because the client needs to keep continuous data communication with server for live weather data. If the client can request the data from Internet resource directly based on the key the server returned, the network load will be much less. In our proof of concept, we choose Yahoo weather service as the resource. Yahoo weather widget can be activated by setting the WOEID (Where On Earth Identifiers) which is unique and non-repetitive 32-bit identifier that assigns to Yahoo geographic resource. We loaded the WOEID in our own system database first. Then, the server receive the client request for current weather. It invokes the geolocation service to get the formatted address based on the user's current location coordinates. Using the formatted address, server can get the right WOEID from the database and return to client.

In the client, we integrate the Yahoo weather widget to our client by using jquery-plugin named zWeatherFeed. Figure 4.16 shows the result of this plugin. It provides easy access interface, and can keep update weather information continuously.

4.5 Summary

This chapter discussed the multimedia services of our system. We showed several services which provide users audio, weather, and news information. We also illustrated how our services differ from current multimedia applications by using the Internet data.

[3]http://forecast.io/
[4]http://www.worldweatheronline.com/
[5]http://www.wunderground.com/

Figure 4.15: The example of the weather forecast

Figure 4.16: The example of the zWeatherFeed

Chapter 5

Experiments

In this chapter, we will verify our proposed system by three experiments. We will first introduce the methods of measurements of this system. Then we will illustrate the design for each experiment. At last, we give the experiment result and conclusions.

We analyze the performance of the proposed work under two different perspectives: 1) The optimization of feedback data from the mashup server. 2) The consumption of network for transiting data. The first perspective is to evaluate the algorithms we implemented in the mashup server. We need to verify whether the result from our server is better than the one from the Internet or not. In other words, we want to figure out the quality of the data that processed by our algorithms. For second perspective, since the server is assumed to be established in the cloud under VANET, the communication approach is mostly wireless, we need to consider the network connection stability including bandwidth stability and package loss. Our infotainment system includes some applications that needs continuous connection with VANET such as navigation and weather functions. Thus, we want to evaluate the enhancement of the data transition between the server and the client. We have two experiments to display the performance of our system. Since there is no existed VANET so far, we consider the network environment as in the Internet. We assume that the clients are under good network connection which means the vehicles are in the urban area referring as service area. And the client is equipped with GPS device. We establish the mashup server on PC. The PC information: Intel(R) Core(TM) i7-3720QM @2.60HZ, OS: Windows 8.1 Pro, IDE : Visual Studio 2012. There are two wireless network interface cards, wired NIC connects to the Internet while the wireless one is considered as the network interface towards the clients. For the client, we developed an Android application on a tablet. We use Juqery framework and phonegap framework to develop the Android application, MSSQL as the local database. Figure 5.1 shows the test page of the services in the PC. For the experiments, we published the webservices to the Internet working as middleware server. The host address is `http://uottawainfotainment.somee.com/infotainment.asmx`.

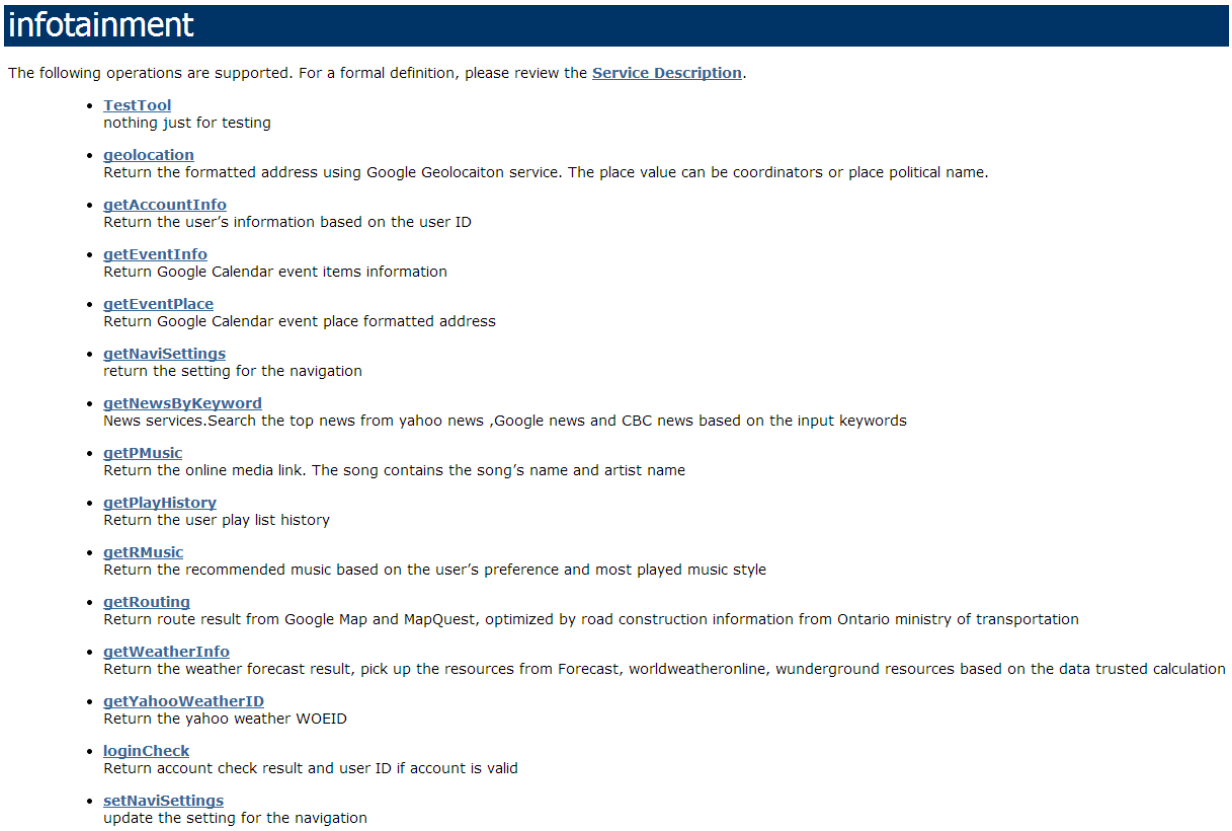


Figure 5.1: Test page of the Web services

5.1 Framework Experiment

One efficient approach to reduce the network data transmission load is to decrease the data size from the server. To drop the abundant data can achieve this goal fairly. What is more, reducing the data size also accelerates the speed of data process effectively.

In our implementation of the server, we have a data filter layer which maintains removing the elements that our server does not need. For example, a weather complete feedback contains a lot of data such as windspeed, pressure, heat, etc. Since we just take the temperature into consideration in our weather service, these data is never involved in the data processing layer. Abandoning these data will reduce the data size of the final feedback without influencing the data processing. Besides, the data fusion can also reduce the data size to a certain extent. For instance, the data fusion for news service will generate the latest news that are related to users' preferences and keywords. For the detail of the work flow of system, we demonstrate news service as an example. We have implemented one web service that searchD the news by keyword.

We use Yahoo pipe, an online data mashup tool, to integrate the news data from Google News, Yahoo News and CBC news. As we explained before, we have no complex data processing in the news service.

In the pipe, we first get the news by keyword from different resources. For Google News and Yahoo News, we receive news through the rss. The query for google news is `http://news.google.com/news?hl =en&ned=us&q=keyword&ie= UTF-8&scoring=d&output=rss"`, and query for yahoo news is `http://news.search.yahoo.com /news/ rss?p=keyword &c=&ei=UTF-8& datesort=1&eo=UTF-8"`. Since there is search query or API for CBC news, we just manually generate a rss news file consisting of the latest news in most fields such as politics, business, health, arts, etc. Next step, we filter the news which contains the keyword. Figure 5.2 is screenshot from the Yahoo pipe console. It is the schematic diagram of news services.

Once the pipe feed is published, any device can get access to this resources from url: `http://pipes.yahoo.com/pipes/pipe.run?_id=029120fb4913ab85dcfdb104759b698d&_render=rss&textinput1=keywords`. But there is still some abundant data in the feedback. In the server, we filter the data furthermore based on our system requirements. In this case, we just need news title, news link, description and published time. So we create a news object that has these four parameters in the system model. Once we receive the data from Yahoo pipe, we instantiate an object using this data. This way we just have the needed data to enter the next layer in the system. We validate framework from two aspects:

1. First, we validate framework by the data size. Since the data filter will drop the abundant data, there should be difference between original file and filtered file. As shown in figure 5.3, we search the news with same keywords. We set the feedback

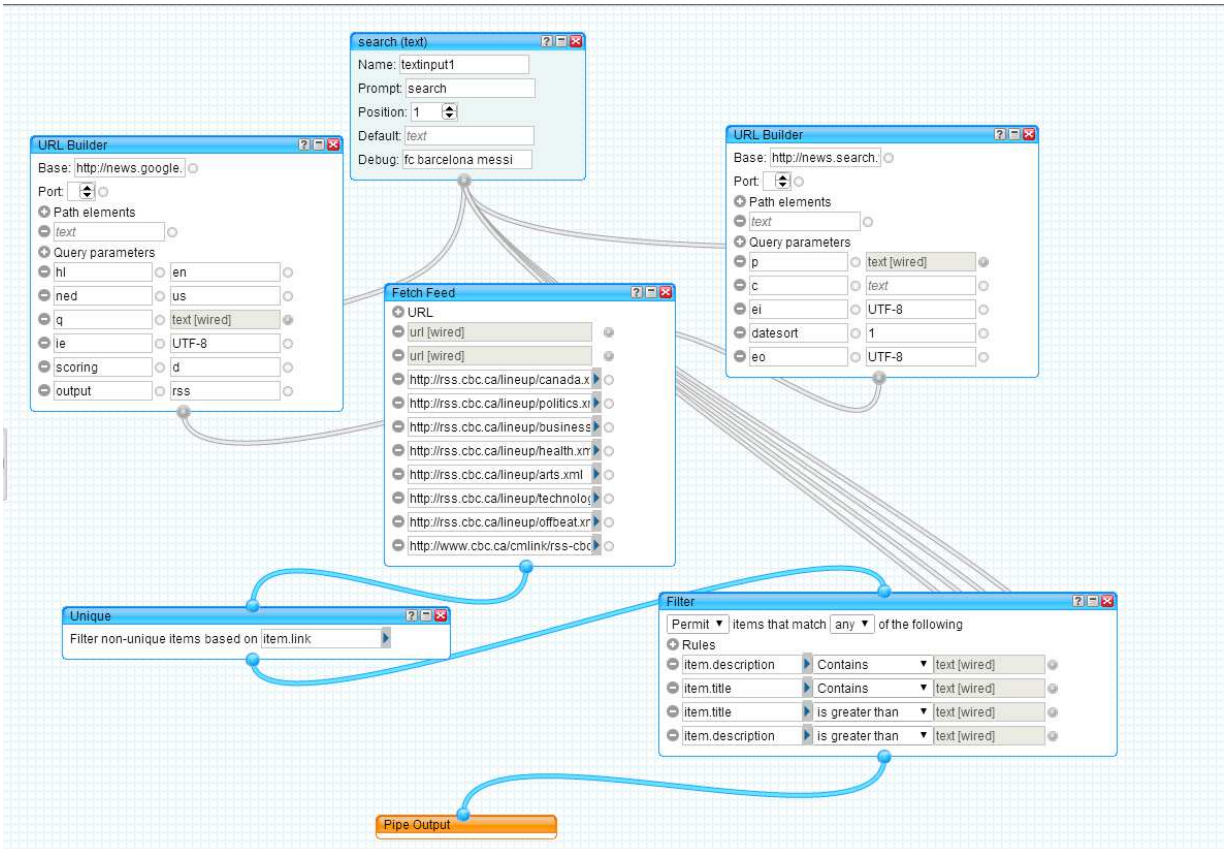


Figure 5.2: Schematic diagram of news service

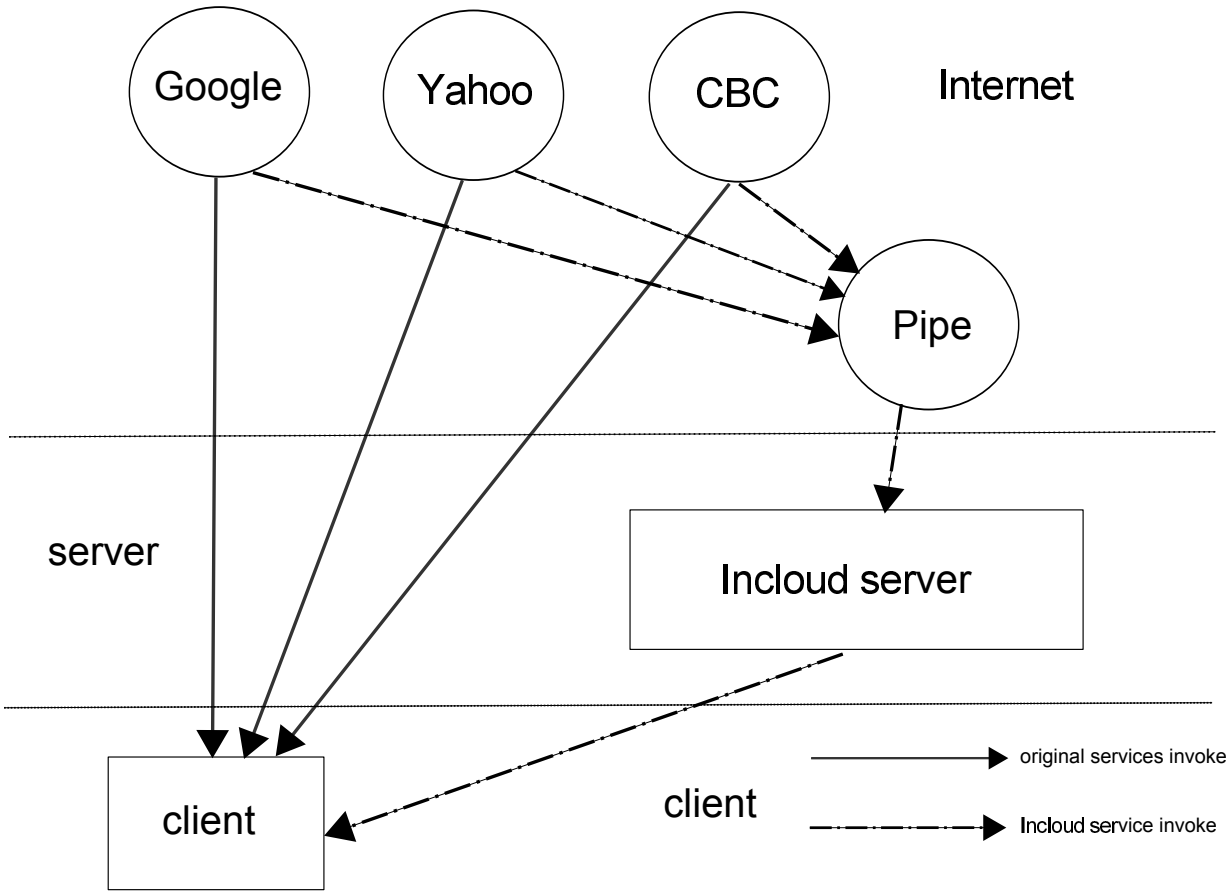


Figure 5.3: Schematic diagram of framework experiment I

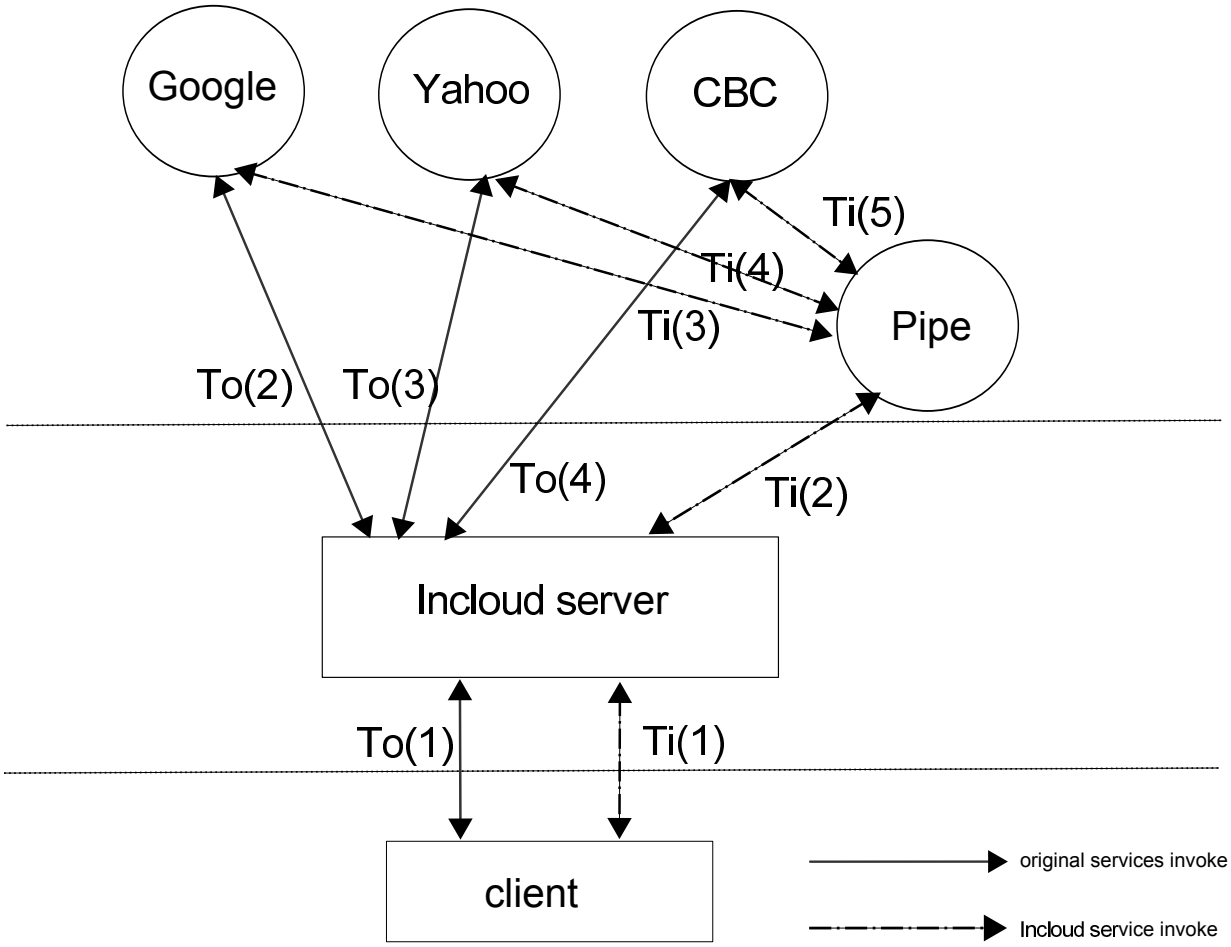


Figure 5.4: Schematic diagram of framework experiment II

from Google, Yahoo, and CBC as f_1, f_2, and f_3. They are directly invoked from Internet resources. Then we make the same search from Incloud which invokes pipe to get result. We set the result as f_i. Next, we put the feedback into a text file and get the file size on the disk. Finally, we compare $f_1 + f_2 + f_3$ and f_i.

2. Second, we validate framework by the transmission time. Considering bandwidth and stability, our connection between client and server is through LAN which is different from VANET. And the framework is in the server as shown in figure 5.4, there is no meaning for us to measure this part time consumption. We make the same search both from three resources and pipe. We can see from the figure 5.4, the time consumption of original search $T_o = T_o(1) + T_o(2) + T_o(3) + T_o(4)$. The time consumption of Incloud search $T_i = T_i(1) + T_i(2) + T_i(3) + T_i(4) + T_i(5)$. The $T_i(n)$ or $T_o(n)$ is the sum of request time and response time. At last, we compare T_i and T_o.

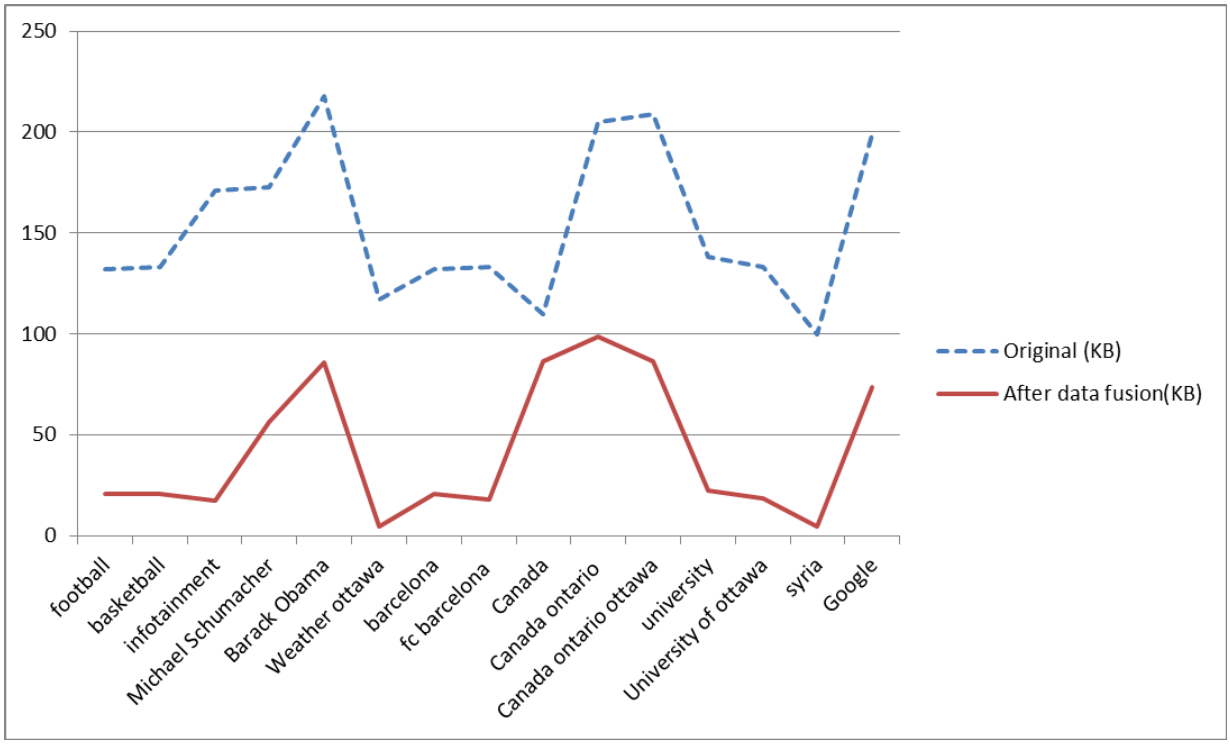


Figure 5.5: The comparison of data size.

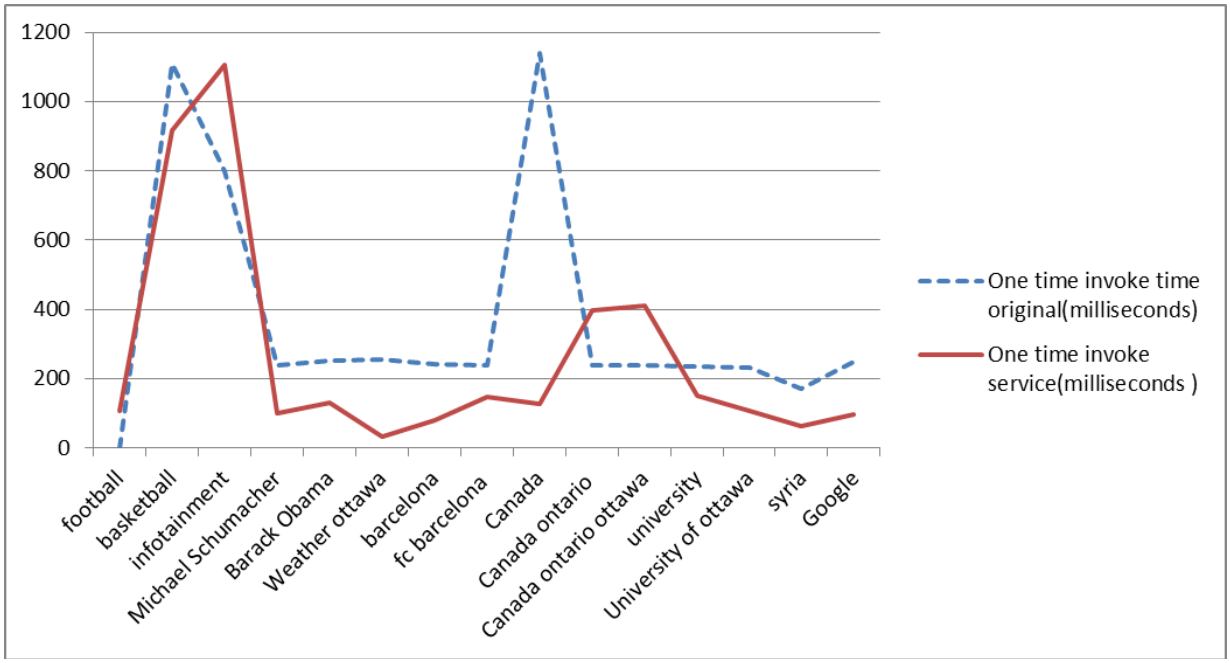


Figure 5.6: The comparison of time consume of data transmission.

Figure 5.5 shows the comparison of the data size between original news data size and the size which is after the data fusion. Figure 5.6 shows the time consumption of data transmission. We take 15 keywords in the experiments and collect the data of result's size difference and transmission time. Among the 15 keywords, we also add a small test for the influence of multiple keywords. The Dashed line represents the file size and time consume directly from the Internet. The solid one represents the result which is after the server data fusion. We can see that the file size has obvious reduction comparing with the original result. And the time consumption appears to be less than 200 milliseconds than original one. Concerning the difference of the search result and complexity of data fusion, the time consumption may be unsteady in some certain cases. This is because different popularities of keywords will lead to different amount of feedback.

5.2 Data Fusion Experiment

Of course, data fusion not only can resize the data package, but also can provide optimized service result based on the multiple resources. The routing service in this implementation reflects this point comprehensively.

In our implementation, the route calculation service is processed based on two map resources: Google Map and MapQuest. For the route optimization, we add two more road construction resources: Ontario Ministry of Transportation and City Of Ottawa. We analyze the data file from these two website: `http://traffic.ottawa.ca/map/construction_list` and `http://www.mto.gov.on.ca/kml/construction.kml`. First, we invoke the geolocation service to get the coordinators of the destination and origin. This service provides the formatted address that can identify unique geographic location on the earth. It also supports the conversion of formatted address like (75 Laurier Avenue East Ottawa, ON K1N 6N5) and geographic coordinates like (latitude: 45.421212 and longitude: -75.679797). We implement this service using the Google Geocoding service API. The request query is : `http://maps.google.com/maps/api/geocode/json ?address="+address+"&sensor=false` and `http://maps.google.com/maps/api/geocode/json?latlng= "+latitude+","+longitude+"&sensor=false`. The service function may decide which query to use by analyzing the request parameters. Second, the system will invoke the routing method after receiving the coordinators of the destination and origin. To get basic route result, we use direction service API from Google Map and MapQuest. The Google Map request query is `http://maps.google.com/maps/api/directions/json?origin="+origin_latitude+","+origin_longitude+"&destination="+destination_latitude+","+destination_longitude+"&sensor=false&alternatives=true`. The MapQuest query is `http://www.mapquestapi.com/directions/v1/route?key=APIKey&callback=renderAdvancedNarrative&avoidTimedConditions=false&doReverseGeocode=false&outFormat=xml&routeType=fastest&timeType=1&enhancedNarrative=falseshapeFormat=raw&generalize=0&unit=k&from="+startWayPoint.latitude+","+startWayPoint.longitude+"&to="`

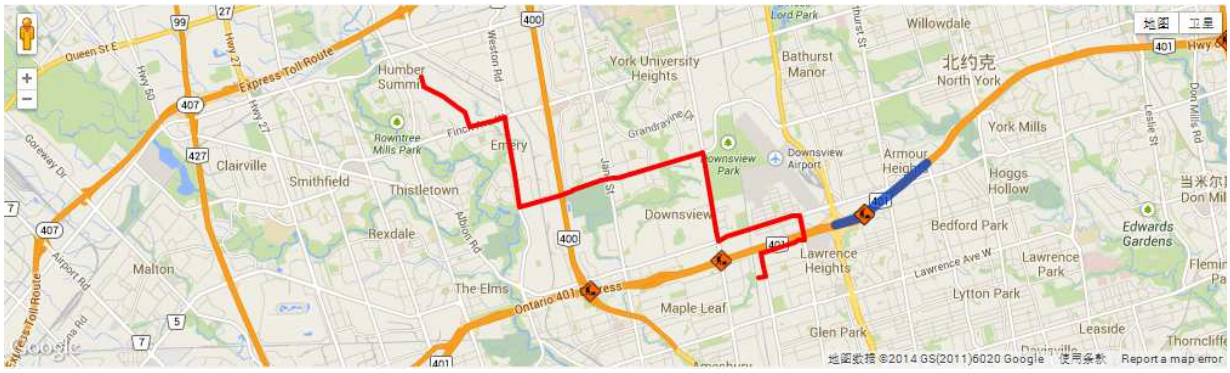


Figure 5.7: The result example of eDirection

`+endWayPoint.latitude+","+endWayPoint.longitude+"&drivingStyle=2&highwayEfficiency=21.0`. We set Google Map result as the main route resource due to reputation rank in the implementation. Next, we optimize the route by recalculating the route to avoid the road construction from the road construction resources. In the client, we implement an android application to simulate vehicle navigation system. We use Google Map to display the route visually. We use JQuery and Google Map Javascript library to receive the data and draw the polyline on the map as shown in figure 5.7.

We designed an experiment that compares our routing result with Google Map. We display both routes for the same origin, destination and distance. Since our route may not the same as the Google Map result due to the road construction information, we also try to calculate the detour road distance. We picked up 10 routes in Ottawa. Figures 5.8 and 5.9 illustrate the result, the red route is the result from our server, and blue route is from Google Map. The marker means the road construction point. We observe that see our system provides the user different routes in order to avoid constructions. From the figures we can find the distance of our result is longer than Google Map. This is because we choose the shortest way option for Google Map. But the driver need longer driving time to bypass the construction to destination. Table 5.1 gives the distance data of these 10 routes. We figure out if we add detour distance,our result is a little shorter. Because most users won't be able to find reasonable detour road due to lacking knowledge of Ottawa Road Map. This optimization is more obvious if the user is a new comer to Ottawa.

Webservices are designed mainly to share heterogeneous data. Our system keeps the same principle to provide users with more valuable data through less interfaces. In this experiment, we measure the efficiency of our map routing application. Our map routing application follows the system data fusion method and we compare the resulting distance with the map routes generated by Google or MapQuest. We have empirically selected ten different source to destination routes as input to the routing application and Google or MapQuest. All the routes are from City of Ottawa, Canada where some road constructions are ongoing and there geolocations are reported in the Ontario Transport Ministry web. Figure 5.10 analyzes the result from figures 5.8 and 5.9. It illustrates that our

(a) (b)

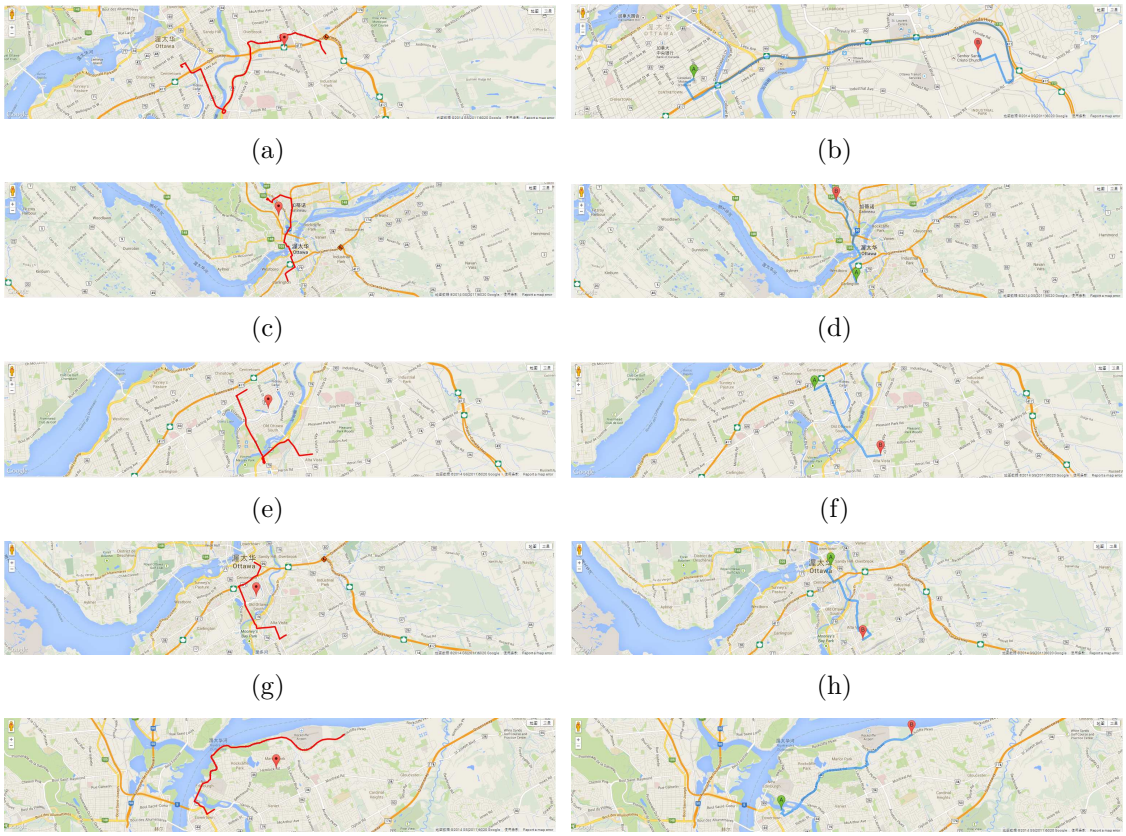
(c) (d)

(e) (f)

(g) (h)

(i) (j)

Figure 5.8: Route experiments part I



(a) (b)

(c) (d)

(e) (f)

(g) (h)

(i) (j)

Figure 5.9: Route experiments part II

Table 5.1: Route distance result comparison

Coordinate Geometry	**Incloud result(KM)**	**Google map + detour distance(KM)**
Origin: 45.420203, -75.678305 Destination: 45.413093, -75.653843	6	7.1
Origin: 45.473614, -75.738228 Destination: 45.422191, -75.681236	10.27	10.9
Origin: 45.32922, -75.515656 Destination: 45.345993, -75.442785	9.8	11.1
Origin: 45.41243, -75.692394 Destination: 45.369152, -75.634201	12.2	13.8
Origin: 45.413756, -75.687832 Destination: 45.419359, -75.632385	8.3	10
Origin: 45.413153, -75.68869 Destination: 45.418034, -75.620112	8.9	16
Origin: 45.379402, -75.71883 Destination: 45.486493, -75.759857	18.937	23.9
Origin: 45.404235, -75.695464 Destination: 45.383682, -75.664572	5.9	6.3
Origin: 45.420986, -75.687416 Destination: 45.420986, -75.687416	8.8	9.6
Origin: 45.436767, -75.687931 Destination: 45.463502, -75.624073	8.6	10.9

routing system offered more efficient route in terms of distance. This is because neither Google or MapQuest considers the current construction data. As a result, when one user follows Google/MapQuest recommended routes, they end up with road blocks and eventually costly path search. While we employ data fusion of Google, MapQuest and Ontario Transport Authority Web and find the best up-to-date low cost route for the vehicle user.

5.3 Context aware Experiment

Our system also considers the context-ware features. We create several services which provides users context aware results. Our system stores the users' profiles, system analyzes users' operations to get users' preferences. Combining with some other information like GPS data and weather information, our system gives better locality-based services like route, music.

We have two different approaches to reflect context-aware features. The first is to put the context-aware elements into consideration during the service call. We take music service

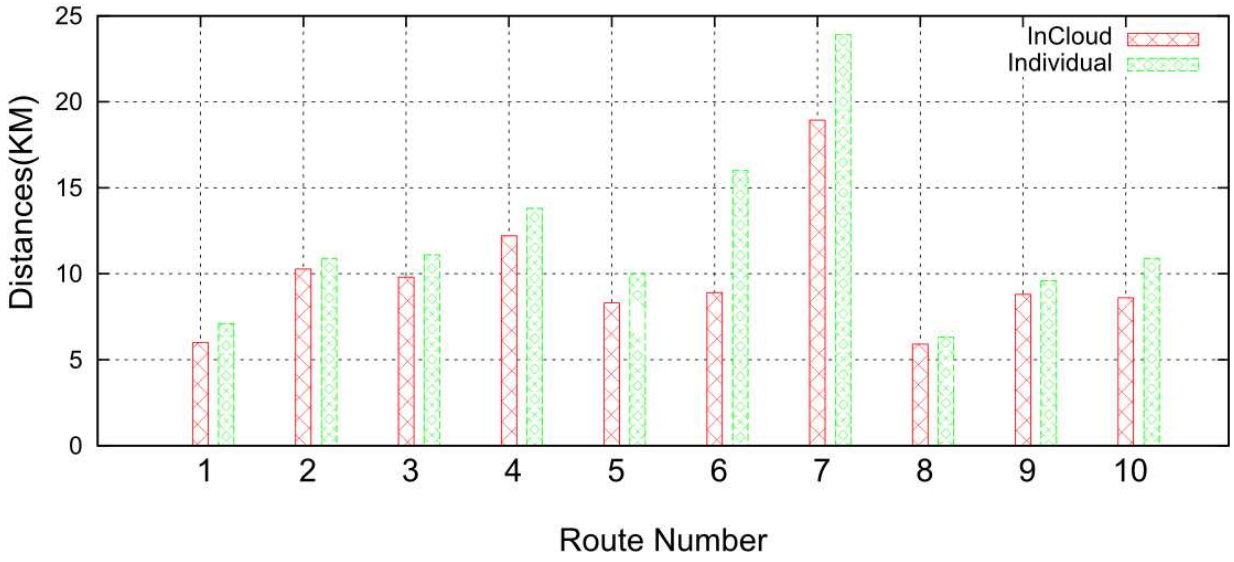


Figure 5.10: *eDirection* application routing efficiency result.

as example. GetRMusic is one music service which provides the user with recommended music based on the users' play history and preferences. To test the user experience, we have requested fifteen users (10 male, 5 female) to participate in the user study. For the test, each user queries ten songs (song name, artist name) in the test user interface one by one. The interface displays two sets of recommendations, one from the lastfm and the other following our fusion method. Details of the application name and the publisher name are hidden to avoid the user's bias. Later we have provided questionnaires to the users and asked them to rate both the recommendation result in 1-5 scale where 1-strongly disagree, 2-disagree, 3-neutral, 4-agree, 5-strongly agree. Table 5.2 shows the average score of the five questions. From the result of the user experience survey, the users favored our recommendation over the lastfm solution. A probable reason is that our recommendation system aggregates results from multiple sources and finally provides a fused one.

The other one approach is that we make composition services to provide users complex service. We will explain it using a scenario: Assuming a user is driving on the road. Our system may connect to user's calendar through calendar service. It can search the coming events in the calendar and get the event time and event place. Then it invokes the route service to get the route from current place to the destination. What is more, the system can estimate the travel time based on the route information and weather condition. The weather condition can be gathered through weather service. At last, the system will give the user a notice or an alert to inform of the user's schedule. During this case, the user does not need to do any operation except login the system. The system will provide user reasonable notifications based on his or her calendar information, current position, weather condition and road condition. It composes the service to generate a new complex service in certain order as depicted in figure 5.11.

Table 5.2: User study of the personalized music recommendation application

Question	**lastfm**	**InCloud**
I get effective recommendations from this system	3.3	4.2
I am more likely to listen to the recommended music	3.6	4.1
I am willing to use the recommendation system	3.3	3.9
I would prefer the music recommendations for my car infotainment system	3.9	4.3
Overall, I am satisfied with the system	3.4	4.0

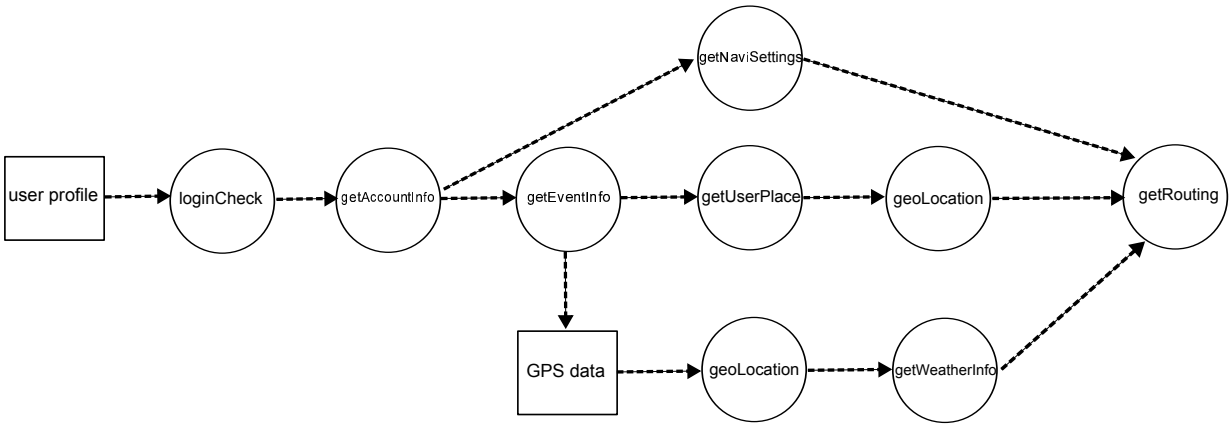


Figure 5.11: The schematic diagram of service composition

Services composition also reflects re-usability feature. Based on the example of service composition, we can see that the more reusable services are created, the more operations will be completed in the cloud, which inherently reduces the client workload.

5.4 Summary

For the experiments part, we designed three experiments to verify our system from different perspectives. We have one news experiments for data size reduction and formation unification of our framework. We also have route experiment and multimedia experiment to reflect the data fusion feature of our system. From the result we can see, our system has significant improvements in these features.

Chapter 6

Conclusion and Future work

With the development of VANET, the vehicles are not just transportation tool any more but also a place where users can enjoy multimedia information and social information. What is more, the access to VANET makes the vehicles available to the Internet. This may further enlarge the capabilities of the vehicles if the vehicles have proper application to use these data. Motivated by Cloud computing and mashup services, we have proposed a cloud based middleware framework for vehicular infotainment application development. The framework follows four design principles: data fusion, context-awareness, re-usability, and loose-coupling. In the framework, all the services are either hosted in the Road Side Unit or in the common cloud. Services consume the Internet resources and compile the solution and send back as little operation intensive data as possible to the application client. Application client only takes input from the user and try to display the result with as little operation as possible. In order to realize the effectiveness of the proposed framework, we have developed many services and used them to build three applications: *eDirection* map routing, personalized music recommender, and context-aware news. In the experimental results, we employ the applications and both quantitatively and qualitatively measure their efficacy in fulfilling the design principles. The experimental results portray that the proposed middleware framework is an effective and efficient method for developing vehicular infotainment applications.

For the future work, we expect to improve our system from following perspectives:

1. **Enlarge scope of the resource :** Currently, we just manually choose the Internet resource based on reputation credit. Next, we can extend a system module that can dynamically select the web resources. Since our system gets access to Internet through VANET, we need to consider several elements such as resource server reaction time, server data transmission loss, and the resource value for choosing a better resource. Because these elements may change when the vehicles moving in the VANET, we need to design a mechanism for dynamical resource choosing.

2. **Enhance user communication with server :** In the prototype, the client is just to display the data rarely update useful information to server. Next step, we hope we can enhance it. For instance, we can design live traffic report mechanism. The user of our system not only can get the road condition online, but also they can report their traffic information in their trips to the server. Thus, our server will have most updated information to other users.

3. **Improve the interface of the client :** The interface in our current client interface is designed based on Nuha's work which has good icon design but lacks of menu organization such as menu items design, user interface for data interaction.

We envision to develop more applications following the framework such as context-aware weather, need based hotel-restaurant recommender, price efficient gas station recommender, etc. Also, we target to build a working VANET model which can be leveraged to build various other applications. Additionally, we want to improve the data forwarding component in our implementations to support more than one RSU and application hand over scenarios.

References

[1] Herrero Agustin, José Luis, Pablo Carmona, and Fabiola Lucio. A model-driven approach for service oriented web 2.0 mashup development. In *ICIW 2013, The Eighth International Conference on Internet and Web Applications and Services*, pages 246–251, 2013.

[2] Tarik Al-Ani. *Android In-Vehicle Infotainment System (AIVI)*. PhD thesis, University of Otago, 2012.

[3] Anwar Aldris, Ariadi Nugroho, Patricia Lago, and Joost Visser. Measuring the degree of service orientation in proprietary soa systems. In *SOSE*, pages 233–244, 2013.

[4] Marica Amadeo, Claudia Campolo, and Antonella Molinaro. Enhancing ieee 802.11 p/wave to provide infotainment applications in vanets. *Ad Hoc Networks*, 10(2):253–269, 2012.

[5] Justin Berkowitz. The history of car radios. `http://www.caranddriver.com/features/the-history-of-car-radios`, 2010.

[6] Peter F Brown, Rebekah Metz, and Booz Allen Hamilton. Reference model for service oriented architecture 1.0. Technical report, Tech. rep., 2006. Retrieved December 16, 2008 from http://docs. oasis-open. org/soa-rm/v1. 0/soa-rm. pdf, 2005.

[7] SIMON Buckingham. What is gprs. `http://www.gsmworld.com/technology/gprs/intro.shtml`, 2000.

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

[9] Ho Ting Cheng, Hangguan Shan, and Weihua Zhuang. Infotainment and road safety service support in vehicular networking: From a communication perspective. *Mechanical Systems and Signal Processing*, 25(6):2020–2038, 2011.

[10] Jan Dannenberg and Christian Kleinhans. The coming age of collaboration in the automotive industry. *Mercer Management Journal*, 17:88–94, 2004.

[11] Yaser P Fallah, Ching-Ling Huang, Raja Sengupta, and Hariharan Krishnan. Analysis of information dissemination in vehicular ad-hoc networks with application to cooperative vehicle safety systems. *IEEE Transactions on, Vehicular Technology*, 60(1):233–247, 2011.

[12] Nuha Feraq. *VANET's Infotainment Services Portal Design.* PhD thesis, University of Ottawa, 2012.

[13] Andreas Festag, Roberto Baldessari, Wenhui Zhang, Long Le, Amardeo Sarma, and Masatoshi Fukukawa. Car-2-x communication for safety and infotainment in europe. *NEC Technical Journal*, 3(1):21–26, 2008.

[14] Darlene Fichter. What is a mashup. *ENGARD, N. Library Mashups Exploring new ways to deliver library data. Information Today*, 2010.

[15] Renato Filjar, Mico Dujak, Boris Drilo, and Dinko aric. Intelligent transport system. *Coordinates*, 5(6):8–10, 2009.

[16] Sandro Rodriguez Garzon. Intelligent in-car-infotainment system: A prototypical implementation. pages 371–374, June 2012.

[17] Maurizio Gibin, Alex Singleton, Richard Milton, Pablo Mateos, and Paul Longley. An exploratory cartographic visualisation of london through the google maps api. *Applied Spatial Analysis and Policy*, 1(2):85–97, 2008.

[18] Kai Hackbarth. Osgiservice-delivery-platform for car telematics and infotainment systems. In *Advanced Microsystems for Automotive Applications 2003*, pages 497–507. Springer, 2003.

[19] Sherin Abdel Hamid, Hossam S Hassanein, and Glen Takahara. Introduction to wireless multi-hop networks. In *Routing for Wireless Multi-Hop Networks*, pages 1–9. Springer, 2013.

[20] Hannes Hartenstein and Kenneth P Laberteaux. A tutorial survey on vehicular ad hoc networks. *Communications Magazine, IEEE*, 46(6):164–171, 2008.

[21] Roy Chaoming Hsu and Liang-Rui Chen. An integrated embedded system architecture for in-vehicle telematics and infotainment system. volume 4, pages 1409–1414. IEEE, 2005.

[22] Chenn-Jung Huang, Chin-Fa Lin, Ching-Yu Li, Che-Yu Lee, Heng-Ming Chen, Hung-Yen Shen, You-Jia Chen, and I-Fan Chen. Service-oriented routing and clustering strategies for vehicle infotainment dissemination. *Int. J. Innov. Comput. Inf. Control*, 7(3):1467–1480, 2011.

[23] Rasheed Hussain, Fizza Abbas, Junggab Son, and Heekuck Oh. Tiaas: Secure cloud-assisted traffic information dissemination in vehicular ad hoc networks. In *Cluster, Cloud and Grid Computing (CCGrid), 2013 13th IEEE/ACM International Symposium on*, pages 178–179. IEEE, 2013.

[24] Rasheed Hussain, Junggab Son, Hasoo Eun, Sangjin Kim, and Heekuck Oh. Rethinking vehicular communications: Merging vanet with cloud computing. pages 606–609. IEEE, 2012.

[25] Ho-Yeon Kim, Young-Hyun Choi, and Tai-Myoung Chung. Rees: Malicious software detection framework for meego-in vehicle infotainment. pages 434–438. IEEE, 2012.

[26] Minyoung Kim, Jung-eun Lee, and Jong-wook Jang. Implementation of the android-based automotive infortainment system for supporting drivers safe driving. In *Ubiquitous Information Technologies and Applications*, pages 501–508. Springer, 2014.

[27] Robin Larsson and Maryam Zarrinjouei. Value creation from an in-vehicle infotainment perspective: A case study. *null*, 2011.

[28] A Lattanzi, F Bettarelli, R Toppi, and E Capucci. Implementing a car infotainment system using nu-tech framework. pages 452–456. IEEE, 2009.

[29] Kunqiong Li, Yuan Fang, Huali Tang, Xiao Liu, and Jishen Liang. A customized services model based on web applications in cloud computing environment. In *Informatics and Management Science IV*, pages 433–438. Springer, 2013.

[30] Chen Liu, Jianwu Wang, and Yanbo Han. Mashroom+: An interactive data mashup approach with uncertainty handling. *Journal of Grid Computing*, pages 1–24, 2013.

[31] Gianpaolo Macario, Marco Torchiano, and Massimo Violante. An in-vehicle infotainment software architecture based on google android. pages 257–260. IEEE, 2009.

[32] Eshed Ohn-Bar, Cuong Tran, and Mohan Trivedi. Hand gesture-based visual user interface for infotainment. In *Proceedings of the 4th International Conference on Automotive User Interfaces and Interactive Vehicular Applications*, pages 111–115. ACM, 2012.

[33] Tim o'Reilly. *What is web 2.0*. O'Reilly, 2009.

[34] Sathyanarayanan Rangarajan, Monica Verma, Anand Kannan, Ayush Sharma, and Ingmar Schoen. V2c: a secure vehicle to cloud framework for virtualized and on-demand service provisioning. In *Proceedings of the International Conference on Advances in Computing, Communications and Informatics*, pages 148–154. ACM, 2012.

[35] Rowdypat. History of car radio retrieved. `http://rowdypat.wordpress.com/2012/06/22/history-of-car-radio/`, 2012.

[36] Pierpaolo Salvo, Francesca Cuomo, and Andrea Baiocchi. Infotainment applications support in vanet. *DIET Department-University of Roma, Via Eudossiana*, 18:00184, 2012.

[37] Pierpaolo Salvo, Francesca Cuomo, Andrea Baiocchi, and Andrea Bragagnini. Road side unit coverage extension for data dissemination in vanets. In *Wireless On-demand Network Systems and Services (WONS)*, pages 47–50. IEEE, 2012.

[38] Pierpaolo Salvo, Mario De Felice, Francesca Cuomo, and Andrea Baiocchi. Infotainment traffic flow dissemination in an urban vanet. In *Global Communications Conference (GLOBECOM)*, pages 67–72. IEEE, 2012.

[39] Christoph Schroth and Till Janner. Web 2.0 and soa: Converging concepts enabling the internet of services. *IT professional*, 9(3):36–41, 2007.

[40] Hemant Sharma, Roger Kuvedu-Libla, and A.K Ramani. Component oriented human machine interface for in-vehicle infotainment applications. In *Proceedings of the World Congress on Engineering*, volume 1. Citeseer, 2008.

[41] Hemant Sharma, Roger Kuvedu-Libla, and A.K Ramani. Confra: A context aware human machine interface framework for in-vehicle infotainment applications. In *Proceedings of the International MultiConference of Engineers and Computer Scientists*, 2008.

[42] Kamal Kumar Sharma, Hemant Sharma, and AK Ramani. mcar: software framework architecture for in-vehicle pervasive multimedia services. In *Proceedings of the International MultiConference of Engineers and Computer Scientists*, volume 1, 2009.

[43] Jan Sonnenberg. Service and user interface transfer from nomadic devices to car infotainment systems. pages 162–165. ACM, 2010.

[44] Toby Velte, Anthony Velte, and Robert Elsenpeter. *Cloud computing, a practical approach*. McGraw-Hill, Inc., 2009.

[45] Yao Wang and Julita Vassileva. Trust and reputation model in peer-to-peer networks. In *Peer-to-Peer Computing, 2003.(P2P 2003). Proceedings. Third International Conference on*, pages 150–157. IEEE, 2003.

[46] Yu Wang and Fan Li. Vehicular ad hoc networks. In *Guide to wireless ad hoc networks*, pages 503–525. Springer, 2009.

[47] Richard Welke, Rudy Hirschheim, and Andrew Schwarz. Service-oriented architecture maturity. *Computer*, 44(2):61–67, 2011.

[48] Md Whaiduzzaman, Mehdi Sookhak, Abdullah Gani, and Rajkumar Buyya. A survey on vehicular cloud computing. *Journal of Network and Computer Applications*, 2013.

[49] Diane Williams. The arbitron national in-car study, 2009.

[50] Jin Yu, Boualem Benatallah, Fabio Casati, and Florian Daniel. Understanding mashup development. *Internet Computing, IEEE*, 12(5):44–52, 2008.