1-1-2007

Mashing up web-based participatory GIS : potential for future e-government services

Jiansheng Gong
Ryerson University

Follow this and additional works at: <http://digitalcommons.ryerson.ca/dissertations>

Part of the [Civil Engineering Commons](#)

Recommended Citation

Gong, Jiansheng, "Mashing up web-based participatory GIS : potential for future e-government services" (2007). *Theses and dissertations*. Paper 260.

NOTE TO USERS

This reproduction is the best copy available.

UMI[®]

61819736X

**MASHING UP WEB-BASED PARTICIPATORY GIS:
POTENTIAL FOR FUTURE E-GOVERNMENT SERVICES**

4
70.212
-G66
2007

by

Jiansheng Gong

B.Eng., Ryerson, 2006

A thesis

presented to Ryerson University

in partial fulfillment of the

requirements for the degree of

Master of Applied Science

in the Program of

Civil Engineering

Toronto, Ontario, Canada, 2007

©Jiansheng Gong, 2007

**PROPERTY OF
RYERSON UNIVERSITY LIBRARY**

UMI Number: EC54187

INFORMATION TO USERS

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleed-through, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

UMI[®]

UMI Microform EC54187
Copyright 2009 by ProQuest LLC
All rights reserved. This microform edition is protected against
unauthorized copying under Title 17, United States Code.

ProQuest LLC
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106-1346

Borrower's Page

Ryerson University requires the signatures of all persons using or photocopying this thesis. Please sign below, and give address and date.

Name of Borrowers	Date	Address	Signature

Mashing up Web-based Participatory GIS: Potential for Future e-Government Services

JianSheng Gong

Master of Applied Science, 2007

Civil Engineering Program

Ryerson University

Abstract

Recent developments of Google Map, Yahoo! Map and MapQuest APIs have pushed map mashups almost to the top among other types of mashups such as news mashups, search and shopping mashups, and video and photo mashups. However, map mashups should not rely on these open data sources and services only. Developing a mashup, in general, implies a number of challenges. This thesis first evaluates existing mashup technologies, such as Google Map, Yahoo! Map, and AOL (MapQuest) in the context of map mashups. The results are then used in creating two map mashups for the two selected case studies related to participatory GIS for e-government services. Several commonly used APIs have been selected and integrated into the prototype mashups to support municipal planning and West Nile reporting services, to assess the potential issues concerning data and technology integration for online participatory GIS. The results demonstrate that map mashup has great potential to improve and facilitate the rapid development of the future participatory GIS and e-government services. However, the existing mashup technologies may not provide techniques or designs satisfying some special e-government requirements.

Acknowledgements

First of all, I would like to express my great appreciation and thanks to my supervisor, Professor Songnian Li, for his valuable advices and ideas as well as constructive suggestions and productive comments on my thesis. I also thank him for providing me the great opportunity to continue my study in the field of GIS and to work with him as a research assistant during my years at Ryerson University. I greatly appreciate his efforts to provide me financial support throughout my course and thesis studies. Without this support, I would never complete this thesis work.

I also attribute my accomplishment to Professor Michael Chapman and other faculty and staff members in the Department of Civil Engineering, for their help and support. Many thanks go to Desmond Rogan for their technical assistance for solving computer and software problems. Many thanks are also extended to Mary Neelands, Kim Kritzer and Dianne Mendonca for their administrative support.

The School of Graduate Studies of Ryerson University is acknowledged for providing me the Ryerson Graduate Awards and the opportunity to conduct my research at Ryerson University from 2006 to 2007.

I also wish to thank my fellow graduate students in the Program of Civil Engineering at Ryerson University, Zheng (Eric) Chang, Nedim Oren, and ChengFeng Xiong for their help. The study time with them always recalls my beautiful and happy recollections.

Last, but not least, I wish to express my gratitude to my parents and Yuan Liu for their unconditional love and support. Without their help and understanding, I would never have completed this thesis.

Table of Contents

AUTHOR'S DECLARATION	II
BORROWER'S PAGE.....	III
ACKNOWLEDGEMENTS	V
TABLE OF CONTENTS	VII
LIST OF FIGURES	IX
LIST OF TABLES.....	X
LIST OF ABBREVIATIONS AND ACRONYMS.....	XI
CHAPTER 1 INTRODUCTION.....	1
1.1 PROBLEMS.....	1
1.2 OBJECTIVES.....	4
1.3 METHODOLOGY	5
1.4 SCOPE AND LIMITATIONS.....	6
1.5 THESIS ORGANIZATION	7
CHAPTER 2 BACKGROUND	8
2.1 WHAT IS MAP MASHUP?	8
2.2 RELATED TECHNOLOGIES.....	12
2.3 MASHUP FOR PARTICIPATORY GIS	22
CHAPTER 3 EVALUATION OF MAP DATA SOURCES AND APIS	24
3.1 EVALUATION CRITERIA	24
3.2 GOOGLE MAP AND APIS	26
3.2.1 <i>Introduction</i>	26
3.2.2 <i>Evaluation</i>	27
3.3 YAHOO! MAP AND APIS.....	30
3.3.1 <i>Introduction</i>	30
3.3.2 <i>Evaluation</i>	30
3.4 MAPQUEST APIS	34
3.4.1 <i>Introduction</i>	34
3.4.2 <i>Evaluation</i>	35
3.5 OTHER MAP DATA SOURCES AND APIS.....	37
3.6 NON-MAP DATA APIS.....	39
3.6.1 <i>YouTube and YouTube API</i>	39
3.6.2 <i>Flickr and Flickr API</i>	40
3.6.3 <i>Evaluation</i>	42
3.7 SUMMARY	43
CHAPTER 4 CASE STUDIES: MASHING UP SELECTED PGIS	47
4.1 PPGIS FOR MUNICIPAL PLANNING SUPPORT	47
4.1.1 <i>Requirements of Mashup</i>	47

4.1.2 Mashup Prototype.....49

4.2 WEST NILE REPORTING SERVICES.....66

4.2.1 Requirements for Mashup.....66

4.2.2 Mashup Prototype.....67

4.3 STUDY RESULTS.....77

4.3.1 Advancements.....77

4.3.2 Limitations.....79

CHAPTER 5 CONCLUDING REMARKS81

5.1 REVIEW OF THE THESIS OBJECTIVES.....81

5.2 OBSERVATIONS AND RECOMMENDATIONS.....83

REFERENCES86

APPENDIX88

APPENDIX A: SAMPLE WEST NILE VIRUS SURVEILLANCE FORMS.....89

APPENDIX B: CODE SEGMENTS FOR STUDY CASE ONE.....96

APPENDIX C: CODE SEGMENTS FOR STUDY CASE TWO99

List of Figures

FIGURE 2.1 GENERAL MASHUP ARCHITECTURE.....	10
FIGURE 2.2 THREE-TIER MASHUP ARCHITECTURE USING ORACLE SPATIAL (RUBIO, 2007).....	11
FIGURE 2.3 WEB 2.0 MEME MAP SHOWING THE WORK PROCESS (O'RAILLY, 2005)	14
FIGURE 2.4 THE SYNCHRONOUS INTERACTION PATTERN OF A TRADITIONAL WEB APPLICATION (TOP) COMPARED WITH THE ASYNCHRONOUS PATTERN OF AN AJAX APPLICATION (BOTTOM) (GARRETT, 2005)	17
FIGURE 2.5 SAMPLE CODE FOR A WEB PAGE CONTAINING RSS FEEDS.....	21
FIGURE 2.6 STATISTICS OF VARIOUS MASHUPS.....	22
FIGURE 3.1 GOOGLE MAP USER INTERFACE.....	27
FIGURE 3.2 BIRD'S EYE VIEW	38
FIGURE 3.3 THE VIDEO PLAYED ON YOUTUBE	40
FIGURE 3.4 FLICKR PHOTO DISPLAYED WITH NOTES	41
FIGURE 3.5 SAMPLE VIEWS FROM DIFFERENT MAP DATA SOURCES.....	44
FIGURE 4.1 MUNICIPAL ENVIRONMENTAL ASSESSMENT PROCESS (AFTER [OGRA, 2000])	48
FIGURE 4.2 USER'S VIEW OF THE FUNCTIONAL DESIGN	50
FIGURE 4.3 PARTICIPATION PROCESS CONTAINING DIFFERENT PATHS.....	51
FIGURE 4.4 INFORMATION AND DATA FLOWCHART.....	52
FIGURE 4.5 INTERFACE FOR PROJECT MANAGEMENT	53
FIGURE 4.6 PROJECT MANAGEMENT: BUFFERING	54
FIGURE 4.7 PROJECT MANAGEMENT: CREATE A NEW POINT PROJECT	55
FIGURE 4.8 PROJECT MANAGEMENT: CREATE A NEW LINE OR AREA PROJECT	56
FIGURE 4.9 UPLOADING VIDEO	56
FIGURE 4.10 INTERFACE FOR THE PUBLIC	57
FIGURE 4.11 PLAYING VIDEO RELATED TO THE PROJECT	58
FIGURE 4.12 VIEW POSTED MESSAGES RELATED TO THE PROJECT	58
FIGURE 4.13 POST MESSAGE AND OPINION	59
FIGURE 4.14 MASHUP ARCHITECTURE AND INTERACTIONS BETWEEN COMPONENTS.....	61
FIGURE 4.15 INTEGRATING YOUTUBE VIDEO VIA ASP AND PHP.....	65
FIGURE 4.16 WEST NILE REPORTING AND BROWSING FLOWCHART.....	68
FIGURE 4.17 OVERALL INTERFACE STRUCTURES.....	70
FIGURE 4.18 MAIN INTERFACE OF THE WEST NILE REPORTING MASHUP	71
FIGURE 4.19 DEAD BIRD LOCATION CONFIRMATION INTERFACE.....	72
FIGURE 4.20 BIRD TYPE SELECTION INTERFACE.....	72
FIGURE 4.21 FORM-BASED DATA ENTRY INTERFACE USING FLASH TECHNOLOGY.....	73
FIGURE 4.22 MASHUP ARCHITECTURE SHOWING INTERACTIONS BETWEEN COMPONENTS.....	74
FIGURE 4.23 PHOTO UPLOADING PROCESS	76

List of Tables

TABLE 2.1EXAMPLE MAP MASHUPS ON THE INTERNET.....12

TABLE 3.1 EVALUATION CRITERIA FOR ASSESSING APIS AND DATA SOURCES
FOR MASHUP25

TABLE 3.2 FUNCTIONALITY COMPARISON OF DIFFERENT MAP DATA SERVICES
.....45

TABLE 3.3 COMPARISON OF COMMUNICATION PROTOCOLS SUPPORTED BY
APIS46

TABLE 4.1 STORAGE STRUCTURE TABLE FOR PLACE AND ROUTE63

List of Abbreviations and Acronyms

API	Application Programming Interface
ASP	Active Server Pages
AJAX	Asynchronous JavaScript and XML
CSS	Cascading Style Sheets
CGI	Common Gateway Interface
CAT	Catalog Service Web
CDN	Content Delivery Network
DHTML	Dynamic Hypertext Markup Language
DOM	Document Object Model
GME	Generic Modeling Environment
GIS	Geographic information system
GPS	Global Positioning System
GML	Geography Markup Language
HTTP	Hypertext Transfer Protocol
HTML	Hypertext Markup Language
JMS	Java Message Service
KML	Keyhole Markup Language
LAMP	Linux-Apache-MySQL-PHP
MCEA	Municipal Class Environmental Assessment
PPGIS	Public Participation Geographical Information Systems
PUT	Programmable Unijunction Transistor
PGIS	Participatory GIS

PHP	Personal Home Page
RSS	Really Simple Syndication
REST	Representational State Transfer
SNS	Social Network Service
SOAP	Simple Object Access Protocol
SQL	Structured Query Language
SARS	Severe Acute Respiratory Syndrome
SOA	Service-Oriented Architecture
SSO	Single Sign-On
URI	Uniform Resource Identifier
WSDL	Web Services Description Language
WMS	Web Map Service
WFS	Web Feature Service
WCS	Web Coverage Service
XML	Extensible Markup Language
XHTML	Extensible HyperText Markup Language

Chapter 1 Introduction

The current freely-available open mapping technologies such as Google Maps, Yahoo! Maps, and MapQuest have provided many new opportunities for including geospatial information into various online applications. One of key benefits of these technologies is their ready-to-use map application programming interfaces (API) and the other types of related APIs which can be easily integrated with them to support online participation and collaboration in public decision making processes. The recent advances in mashup, a way to integrate one or more data or service sources to create new web-based online services, provide new horizon for using these APIs. However, these API technologies are often tied up with data/services provided by the technology providers. This raises a number of questions: 1) how can we integrate the other data with the data freely provided; 2) who controls the data; and 3) how the privacy of the collected data is protected for those applications involving public input for example; and 4) if an official record is required, how we can retain a local storage of the data when part of data services are from other sources. As such, it is necessary to study their suitability to more complicated online applications other than those simply include a map. This is especially true, if the web-based online applications are mashed up using both online free data services and the local data services through service APIs

1.1 Problems

As a new concept of developing user-centric online applications based on Web 2.0, mashup is still in its infancy. There are many problems remaining to be solved, although discussions are spotted everywhere on the Internet and in published articles. While it is

not possible or necessary to address all problems related to mashups and mashup development in one thesis, the following discussions focus on those specifically related to and significant to online map mashups.

It is difficult to discover and integrate legacy and local data into new mashup applications. As a new concept of developing web-based data integration applications, mashup is becoming increasingly popular in the Internet. In general, mashup realizes its data integration through logic combination of server computer and client computer. Many mashup applications need to adopt data provided directly by the user. For example, in a GIS-based discussion forum where the posted messages are from forum participants and need to be integrated with map data provided by map API providers. In many such online applications, mashups has at least one dataset coming from users through their local computers. It therefore remains a problem how to collect and integrate the data and resource to be shared through mashup applications and Internet, and how to ensure semantic meaning and quality of integrated data.

For map mashups, people tend to have an impression that mashups must be using Google Maps and the similar applications. There are two issues here. First, map mashups often use maps provided through map APIs as a map-based information “carrier”. These maps provide location data that allow collection and integration of other data to create real-time geo-referenced information. For instance, some map mashups allow us to attach and release information concerning the condition of some specific locations such as photos and videos. Second, in order to create more useful mashups

to a specific application domain or to include some social networking functions, APIs other than map APIs need to be considered and integrated. For example, to realize a function that allows users to post and pinpoint the photos and videos on a map interface, the Flickr or YouTube APIs may be used. However, working with APIs from different sources is not an easy task and requires a better understanding of these APIs as well as a better strategy of application data management. In short, developers of this kind of integrated mashup need to adopt not only map APIs but also other types of APIs. As a result, they will need to look at issues related to data, compatibility between different sets of APIs, and their suitability to a particular type of application.

An important issue in developing general online mashup applications is related to their acceptable use policies which need to be examined more closely (Pietroniro and Fichter, 2007). The online map mashups also hold the same issue, especially when dealing with geospatial data ownership and data privacy protection, for example, the protection of information provided by online participatory GIS users. Mashups, or more generally Web 2.0 paradigm, are, in fact, highly user-centric, meaning that their focus is on enriching users' interactive online experiences by allowing them, not necessarily a professional programmer, to freely create their own things using others' networked resources without getting their prior permission. This raises the biggest social issue facing mashup developers, which is "the trade-off between the protection of intellectual property and consumer privacy versus fair-use and the free flow of information" (Merrill, 2006). Moreover, since the data services stemmed from different, not necessarily

mutually trusting providers, such mashups need to be built on a sound security basis to protection all parties involved.

1.2 Objectives

This thesis project does not intend to address all problems discussed above, although some of them may be included in the relevant discussions. As such, the primary objectives of this thesis are:

1. To conduct selected case studies of online participatory GIS (PGIS) in order to study potential impact of Web 2.0 approaches and mashup technologies on future e-government services development.
2. To evaluate the suitability of the open map APIs and to develop proper guidelines for such online application development, researches and discussions are carried out for the application of Google Map API, Yahoo Map API and the mashups API without maps.

Through two study cases, the first objective is to investigate the possibility of mashing up online applications to support government related services. Particularly, the study will try to find out whether or not current map APIs and other types of APIs as well as the way data services are provided through these APIs are sufficient for such online services. Moreover, the study looks at potential issues such as those related to data management and provides some recommendations.

Built on the evaluation of existing freely-available APIs, the second objective tries to determine the most suitable APIs and mashup development tools for the selected study

cases. This in turn allows further study on some problems described in Section 1.1 and the development of some useful guidelines for developing government fostered map mashups.

1.3 Methodology

A case study approach is adopted in this thesis research. Since online participatory GIS are the main target areas of this research project, two cases were selected at the beginning of the project. One is on prototyping a map-based mashup for West Nile reporting services which has great potential application in three levels of governments in Canada. Another one is on prototyping a GIS-enabled mashup which facilitates the public participation in municipal planning decision-making processes. In developing mashups for the two study cases, a general “how-to” process provided by *programmableweb* (<http://www.programmableweb.com/howto>) is followed, which includes the following steps: Pick a subject, Decide where your data is coming from, Weigh your coding skills, Sign-up for an API, and Start coding. In Chapter 4, this process will be discussed further with respect to the two study cases.

This thesis project has followed the following procedure with the corresponding tasks in each step:

1. ***Literature Review and Mashup Technology Review***: this stage focuses on understanding mashup concept, its development and the technologies involved, especially the APIs that can be used to create different types of mashups;

2. ***Mashup APIs and Data Sources Evaluation***: this step evaluates and reviews selected APIs and data sources as well as mashup building tools that are relevant to the development of participatory GIS;
3. ***Study Case Selection and Requirements Identification***: at this stage parallel to the second step, the study cases are selected and their application requirements are identified and defined for later prototyping work;
4. ***Case Mashup Prototyping***: the prototype mashups of the selected study cases are designed and developed at this stage. The mashups are then assessed to draw some observations.

1.4 Scope and Limitations

Although mashups are intentionally for user-centric web application developments, the available data sources and APIs are overwhelming to master in terms of their documentation, types of programming languages, and time and effort required in creating mashups under these vast amount of sources. Instead of developing mashups for each selected case using different sets of APIs and data sources, or different combinations of them, the project focused on only one combined set of APIs and data sources in creating mashup for each case. This restriction also limits the number of selected study cases for which mashups can be created and tested.

Due to the same reason above and the limited number of study cases selected, it is also not possible to study all aspects that possibly impact the use of mashup technology for creating participatory GIS applications. For example, with respect to the functions provides by selected map APIs, only several specific functions are discussed in this

thesis according to the study case requirements. In addition, this thesis project does not attempt to address social challenges facing mashups although they are very important in their own right.

1.5 Thesis Organization

This thesis is organized in five chapters. In Chapter 1, a general introduction about mashup problems, research objectives, methodology and scope is included. Chapter 2 introduces the concepts of mashup and map mashup, their current development situations concerning this study, and reviews some existing mashups related to participatory GIS. In Chapter 3, relevant data sources and mashup APIs, as the two key components for mashing-up, are reviewed and evaluated. In this chapter, detailed analyses on the current situation of these two key technologies as well as their comparison with several other popular APIs are carried out. Chapter 4 describes the two selected study cases and presents the prototype mashups developed for the cases. Discussions on potential technological incompatibilities and limitations are included. Finally, Chapter 5 presents some concluding remarks, including a summary of research results, discussions of major issues encountered, and suggestions for future work.

Chapter 2 Background

This chapter aims at providing some technical background knowledge on mashup and its development. Since mashup is a fairly new concept in web application domain, it is necessary to establish some basic understanding of its concept, history, current developments and the status of its use in online mapping and the mapping related applications. This background is useful in understanding the technology evaluation in Chapter 3, and mashup prototyping in Chapter 4. In particular; some online participatory GIS using mashup technologies are reviewed.

2.1 What is Map Mashup?

The term mashup is historically rooted in musical style compositions where recordings from different sources are combined into a new piece. For example, in pop music, two songs are remixed and played together resulting in a music mashup. On the Web, it means taking information published from multiple sources and integrating it into a new information stream.

A mashup is a website or web application that uses contents or services from more than one source to create a completely new service (Wikipedia, 2007). Based on Web 2.0, mashups provide exciting new ways of aggregating information services from multiple providers and present them to users over the Internet. A mashup normally consists of some data sources (mostly 2 or more), APIs or RSS feeds to access these data sources, and unified web interfaces created. Mashups can be created by developers, professional programmers or amateurs, using different technologies such as XML, web

services, RSS/Atom, screen scrapping, and Ajax. More details on these related technologies will be discussed in the next section.

We can trace the development of the computer to gain better understanding of the emergence of the mashup. Taking the operating system of our computers as an example, such as Windows, it is the integration of APIs or application program interfaces that help the programmers build their own application programs on top of windows. Computer itself is a good example to provide customer interface. The mouse and keyboard which are familiar to most of us can also be considered as computer interface.

The general architecture of a mashup essentially has three levels or tiers. As illustrated in Figure 2.1 General mashup architecture, the first tier on the bottom contains data sources and services to be used in the mashup. The second tier in the middle holds all mashup applications that bring the data sources and services together for a set of business logics or functions the mashup provides. Finally, the last tier on top has the web user client or interfaces required to access the mashup sites. On this tier, in order to enhance user interaction with the mashup and improve the user experience, advanced web technologies such as JavaScript are used.

Figure 2.1 General mashup architecture

MashupAwards (<http://mashupawards.com/>) has outlined a few important things in general which should be considered before starting to build a mashup. These include: coming up with a simple, useful or practical idea, choosing the right tools and right data sources, and deciding how to host your mashup. There are a number of mashup editing tools available to choose, examples include Yahoo! Pipes, Microsoft Popfly and Google Mashup Editor. At the time of this thesis work, Google Mashup Editor is still in testing stage and only the people who request and are authorized can use it. The selection of data sources is actually to choose related APIs because the data sources can only be accessed through the corresponding set of APIs. Available APIs also include YouTube, Amazon Web Services, Flickr, etc., on top of map APIs from Google, Yahoo and MapQuest.

There are different types of mashups nowadays found in the Internet. These mashups may be categorized as one of the following: map mashups, video and photo mashups, mashups for search and shopping, and news mashups. Mashup has become a phenomenon over the last few years. At the time of this writing, ProgrammableWeb lists around 1,800 known mashup sites and the number is growing rapidly.

Map Mashup

As a special type of mashup, map mashups combine at least one map data source with added information to create a new map (Pietroniro and Fichter, 2007). The architecture of a map mashup follows the same three-tier model, but includes “spatial” elements in all tiers. As an example, Figure 2.2 illustrates the three-tier architecture for map mashups using the Oracle Spatial APIs. For the map mashups, some decisions need to be made in terms of technology selection, e.g., RSS vs. GeoRSS. GeoRSS allows adding of a few tags, similar to HTML tags, to RSS that holds spatial data.

Figure 2.2 Three-tier mashup architecture using Oracle Spatial (Rubio, 2007)

A search on the Internet can easily find hundreds of map mashups created by professional programmers and amateurs for the purpose of either providing geo-

referenced information using a map interface or social networking. Table 2.1 lists some example mashups. More map mashups may be found at *MashupAwards* (mashupawards.com), *programmableweb* (www.programmableweb.com/howto), the corresponding websites about Google Map, Yahoo! Map, and AOL (MapQuest) APIs.

Table 2.1Example map mashups on the Internet

Site Name / URL	Brief Description
Dartmaps http://www.mackers.com/projects/dartmaps/	A real-time tracking of commute trains in Dublin
Frappier http://www.frappier.com/	Social maps: a mash-up of an online guest book, a hit log and a map
GMaps Flight Tracker http://gmapsflighttracker.com/	Track flights by refreshing map display every 40 seconds
HousingMaps http://www.housingmaps.com/	A mashup of Google Map/Earth and craigslist - Local classifieds and forums
Map Builder http://www.mapbuilder.net/	Allows you to tag locations on a map and publish it on your own website
My California Traffic http://www.mycaliforniatraffic.com/	Real-time traffic conditions of highways in California

2.2 Related Technologies

Technologies that are facilitating the development of mashups are overwhelming. It is not feasible to look at all of them in a single thesis. This section briefly introduces and reviews some of the key technologies, identified by Merrill (2007), which have been used or influenced the evolution of mashup and are relevant to this thesis project.

These technologies include Ajax, SOAP and REST, screen scrapping, semantic web and RDF, and RSS and ATOM. Since most of these related technologies are under Web 2.0 paradigm, this section starts with a brief introduction of Web 2.0, followed by review of the above technologies.

Web 2.0

Web 2.0 is a general concept for a new generation of web applications in the Internet comparing with its previous counterpart, Web1.0 (the Internet model before 2003). Transforming from Web1.0 to Web 2.0 is a development from mere “read” to “write” and “build together” , which represents a revolution from the core content to external application (O'Reilly, 2005). As O'Reilly states "Web 2.0 doesn't have a hard boundary, but rather, a gravitational core". He further comments that one "can visualize Web 2.0 as a set of principles and practices that tie together a veritable solar system of sites that demonstrate some or all of those principles" under a web platform at a varying distance from that core. Figure 2.3 illustrates a Web 2.0 meme map that depicts a variety of web applications related to the core principles.

Figure 2.3 Web 2.0 meme map showing the work process (O'Reilly, 2005)

According to Wikipedia (http://en.wikipedia.org/wiki/Web_2), Web 2.0 technology infrastructure includes many complex and evolving technologies such as server-software, content-syndication, messaging-protocols, standards-based browsers with plugins and extensions, and various client-applications. Particularly, web applications based on the Web 2.0 concept use one or more of the following features:

- Rich Internet application techniques, often Ajax-based
- Semantically valid XHTML and HTML markup
- Microformats enriching pages with additional semantics
- Folksonomies (in the form of tags or tagclouds, for example)

- Cascading Style Sheets to separate presentation from content
- REST and/or XML- and/or JSON-based APIs
- Syndication, aggregation and notification of data in RSS or Atom feeds
- Mashups, merging content from different sources, client- and server-side
- Weblog publishing tools
- Wiki or forum software, etc., to support user generated content
- OpenID for transferrable user identity
- Use of Open source software, such as the LAMP stack

While the mashup feature is of primary concern in this thesis, the other features play equally important roles in developing a useful mashup. Some of these feature technologies are described below.

Ajax

Ajax is a web development technique for creating interactive web applications. The key concept of Ajax model is to enable the change of the part of web page content without reloading the display and behaviours of the existing web page. This is realized through asynchronously requesting extra data from the server and loading it into the web page display in background. Rather than being a technology, Ajax is more a web application model (Merrill, 2007), which comprises several technologies primarily including extensible hypertext markup language (XHTML), cascading style sheets (CSS), document object model (DOM), extensible markup language (XML), and server or

browser side scripting languages or technologies such as XMLHttpRequest, IFrame and JavaScript.

Compared with the traditional web application model, a web application based on the Ajax model eliminates the traditional start-stop-start-stop nature of interaction by introducing an intermediary, i.e., an Ajax engine, between the user and the server (see Figure 2.4). The Ajax engine is responsible for both rendering the interface and communicating with the server on the user's behalf (Garrett, 2005). It allows the user's interaction with the application to happen asynchronously, meaning that the user can continue interacting with the application interfaces while the engine is communicating with the server for extra data. In this way, the users of the web application do not need to wait and do nothing while the server is processing user's requests. In addition, the bandwidth consumption for web applications using Ajax technology tends to be cut down.

classic web application model (synchronous)

Ajax web application model (asynchronous)

Jesse James Garrett / adaptivepath.com

Figure 2.4 The synchronous interaction pattern of a traditional web application (top) compared with the asynchronous pattern of an Ajax application (bottom) (Garrett, 2005)

Ajax technologies has been widely used in web applications to provide such functions or usages as form data validation, autocompletion, page loading balance, partial data submission, sophisticated user interfaces, etc. For mashups, Ajax allows the HTML style web pages that include external scripts to access and mix external data or data services.

SOAP

Originally stood for “simple object access protocol” and now “services-oriented access protocol”, the focus of SOAP has been shifted from object-based system to message exchange based interactive operation. The new SOAP provides a protocol for exchanging XML-based messages over computer networks, i.e., a simple XML-based protocol to let applications exchange information over the internet. SOAP has a number of advantage features including platform and language independency, XML-based, simple and extensible, W3C standard, and ability to get around firewalls. However, some of these features may also pose potential security concerns such as firewall passing.

SOAP provides a format of messages sent between Internet/Web applications. The messages are normally encoded as an ordinary XML document that contains the following elements: Envelope, Header, Body, and Fault. While the Envelope and Body elements are required, the other two are optional. The Envelope element is the key to identify the XML documents as a SOAP message; the Fault element provides the

necessary information about errors that occurred while processing the message. All the elements above are declared in the default namespace for the SOAP envelope.

Web services based SOAP APIs are described using web services description language (WSDL) files, including the operations provided by a service and the message format (using XML Schema) they can receive and address. SOAP message is transmitted through HTTP protocol, and also can be transmitted by other ways (such as JMS or E-mail)

REST

The term REST, stands for representational state transfer, was first introduced to describe the Web architecture. Representation State Transfer is intended to evoke an image of how a well-designed Web application behaves: a network of web pages (a virtual state-machine), where the user progresses through an application by selecting links (state transitions), resulting in the next page (representing the next state of the application) being transferred to the user and rendered for their use (Roy, 2002).

REST is closely related to service-oriented architecture (SOA) which is based on the concept of "resource". Everything in REST is considered as "resource", e.g., customers, locations, items. As such, REST web services are SOA based, with a number of limitations such as interfaces limited to HTTP using HTTP GET, DELETE, POST, and PUT methods; and most messages coded in XML.

Generally speaking REST represents a sort of standard operations and communication rules between servers and web pages using XML.

RSS/Atom

RSS is a text-based web content syndication format, often specified using XML. Files related to RSS are normally called RSS files, or RSS feeds or channels. If a website has an RSS feed, it is said to be syndicated. RSS files simply contain a list of items, each of them has a title, summary, and a link of a URL (web address), and optionally other information such as the date and creator's name. In order to read a RSS, software called "RSS reader", "feed reader", or "RSS aggregator" is required. There are two ways in reading RSS: subscribing and entering the RSS feed's link into the RSS reader or using the RSS icon embedded in a browser to start the subscription process. It is the RSS reader's responsibility to check the user's subscribed feeds regularly for content updates. Currently, RSS feeds are most widely used in web news broadcasting and Webblogs.

Atom, on the other hand, is the name of an XML-based Web content and metadata syndication format, quite similar to RSS. However, comparing with RSS, Atom has some major or qualitative differences in terms of deployment, specifications, publishing protocols, required content, payload, autodiscovery, full or partial content, and extraction and aggregation. An in-depth comparison is presented in an article at <http://www.intertwingly.net/wiki/pie/Rss20AndAtom10Compared>.

RSS feeds can be used for displaying headlines from other websites, forming RSS-based search engines, and perhaps, most commonly aggregating RSS. It is simple and easy to create a web page containing RSS feeds, as illustrated in Figure 2.5. However, specialized software is required on the server side to publish a list of feeds or content in a standard format so the RSS reader can understand. RSS/Atom has potential usage in web-based participatory GIS. For example, recent comments from the participants and updates on the projects in public consultation can be pushed to the participants through RSS subscriptions. In addition, mashups allow mapping of GeoRSS or RSS feeds directly onto a map as well as displaying traffic or weather data

```
<?xml version="1.0" ?>
<rss version="2.0">
  <channel>
    <title>RSS/Atom Test Page</title>
    <link>http://www.ryerson.ca/simal/</link>
    <description>A test page for XML User Interfaces...</description>
    <image>
      <url>http://www.ryerson.ca/simal/test.gif</url>
      <link>http://www.ryerson.ca/simal/default.htm</link>
    </image>
    <item>
      <title>Today's News</title>
      <link>http://www.ryerson.ca/simal/test_xml_RSS.htm</link>
      <description>about RSS</description>
    </item>
    <item>
      <title>Upcoming News</title>
      <link>http://www.ryerson.ca/simal/test_xml_rdf.htm</link>
      <description>all about RDF</description>
    </item>
  </channel>
</rss>
```

Figure 2.5 Sample code for a web page containing RSS feeds

2.3 Mashup for Participatory GIS

The major driving force for the success of mashup is the open maps APIs provided by Google, Yahoo and other companies, which just open the doors for Web developers (including the fans, the patch developers and some others) to include all types of data in the maps. Figure 2.6 shows the statistics of various mashups created up to October 12, 2007 (completed by www.programmableweb.com), which include the top 10 categories of mashup websites. As seen in the figure, the map-based mashup accounts for a large proportion (33%) of the all counted mashups (the actual number at the time of this writing is 1377).

Figure 2.6 Statistics of various mashups

It is not a trivial task to classify all map mashups being created so far because of two reasons: 1) the overwhelming number of map mashups available on the Web; and 2) the difficulties of drawing clear boundaries between different map mashups. However, a quick search on the Internet and try-out of those map mashups listed under mapping on www.programmableweb.com website shows that most map mashups are created for applications in real estate (e.g., resale home search), business locations (e.g., beer

stores), transportation and transit mapping (e.g., metro transit map), news incident locations, social networking (e.g., buddy mapping and sharing photos and videos), tourism, crime mapping, etc. Little has been found in providing map-based government services and formal participatory decision making which involve two way communications flow, or mashing up the data sources and services provided by government agencies so that the citizens can carry out their own analysis using lightweight mashups.

Chapter 3 Evaluation of Map Data Sources and APIs

This chapter presents evaluation results of selected map data sources and their APIs widely used for mashing up various web-based spatially-enabled applications. The evaluation results provide a basis for the selection of APIs for prototyping two mashups discussed in Chapter 4. The chapter starts with a brief discussion of the criteria used for the evaluation, followed by detailed discussion of Google Map, Yahoo! Maps, MapQuests data sources and APIs, and other relevant data sources and APIs. Finally, some observations are summarized which are useful for the presentation of case studies in later chapters.

3.1 Evaluation Criteria

A quick literature search finds little on the evaluation criteria used for assessing mashup APIs, tools or platforms for developing mashups. A white paper published by Proto Software describes a set of 17 criteria for evaluating mashup building platforms for business applications (Proto, 2007). These criteria include many closely related to programming level evaluation such as widget level. This thesis aims at providing an evaluation of existing mashup technologies at user level, focusing on some higher level criteria as listed in Table 3.1, and are short-listed and modified from Proto's 17 criteria.

The evaluation criteria are categorized into three groups: data and services, APIs and building tools, and output and deployment. The first group focuses on types of data supported and how data services are provided through corresponding APIs. The second group assesses the involved APIs and their suitability to developing PPGIS and

community GIS. The last group looks at easiness and flexibility of deploying the outcome mashups. These criteria are applied to three brands of widely used map APIs from Google, Yahoo and MapQuest, as well as other APIs related to the thesis work such as YouTube and Flickr.

Table 3.1 Evaluation criteria for assessing APIs and data sources for mashup

Criteria Set	Specific Questions	Code
Mashup Data/Service	What types of data is the mashup platform designed to handle?	C1
	Does the mashup platform work with other data sources?	C2
	Does the mashup platform work with video or audio?	C3
Mashup APIs & Building Tools	Does the mashup platform support integrating with local files and system wide resources?	C4
	Does the mashup platform provide a mashup building environment, e.g., a mashup editor?	C5
Mashup Output & Deployment	What are the overhead costs associated with the mashup development paradigm and its learning curve?	C6
	Can it deploy as a hosted web application?	C7
	Can it deploy to a web server of your choice?	C8
	Can it deploy to a runtime or rich client application?	C9
	What browser or software do end-users need to have installed to access the mashup?	C10

In addition to the above general criteria applied to any kind of mashup, it is also necessary to look at some aspects specific to the creation of map mashups, for example, whether or not the map APIs support the inclusion of geocoding, map annotation, and simple spatial analysis functions in map mashups. This will be examined in evaluating technologies as well.

3.2 *Google Map and APIs*

3.2.1 Introduction

Since Google Map entered the field of providing free online mapping services, it has challenged the colourful map services that used to be solely supplied by MapQuest. One of key features of the Google Map, which makes it outstanding among others, is the integration of satellite images with maps. Google Map provides both map data and APIs to access these data to the public for free. With Google Map, users can choose different ways to display map data, such as map view, bird's eye view satellite images, or a hybrid view of both. Furthermore, it can also provide distances from any starting point to a destination user sets, as well as the best route. In addition, Google Map supports annotations on all views.

A unique feature provided by Google Map is so-called Marker, which allows highlight any particular place with the symbol, adding the description in the associated message box through InfoWindow, creating the route and scope by drawing lines using Polylines and other annotation methods (see Figure 3.1). These services can be easily used in some applications, such as the most commonly seen real estate applications (providing

the geographical information about the houses), job information (providing the local position of enterprises recruiting talent), tourism information (providing scenic spots position and the key landmark), traffic information inquiry (providing the best routes between two places), and adding more colourful image information on the cold address characters. Google Map APIs supports all of the above functions.

Figure 3.1 Google map user interface

3.2.2 Evaluation

Compared with other kinds of map services, Google Map seems to be more popular. It is much easier to understand by most people. Meanwhile, functions provided by Google Map can satisfy the common and ordinary demands of the users. As far as mapping

routes is concerned, Google Map only supports the Reverse Route between the two points; for Actions, Google Map supports Permalink, Email Map, Send to Phone and Printable. As for the services and pricing, Google Map provides free mobile services and APIs, which allows Google Map own a large user community quickly.

1. Mashup Data/Service

Google Map APIs can handle map browsing, ranging, routing services and all map data from Google server for creating map mashups. Google Map APIs only support AJAX-XML data type using XHTML Web Page. They can not work straightforwardly with other data sources. But developer may use ASP or PHP to convert other data type to standard XML for Google Map APIs. For example, to mash up some kind of video/audio API with Google APIs, the developer should build up an extra server to exchange the data between that API and standard XML files. In this way, the Google APIs can get the data in standard XML form.

2. Mashup APIs and Building Tools

Through AJAX technology, Google Map can access local files directly. This is the only way to communicate with other programs or APIs. This encapsulation provides the most security and compatible with other programs, but it restricts the application areas of Google Map APIs.

Google provides a useful Google APIs mashup building environment: Google Mashup Editor (GME). It includes a JavaScript API that gives developers direct access to the

document object model (DOM) via JavaScripts. This API lets the developers use JavaScripts to perform operations that duplicate and go beyond the features available in the GME tags. The API is useful when the developers want to access an object in the application from a JavaScript expression. The developers can also use the API to perform CRUD operations (create, read, update, delete) on entries in a data feed.

3. Mashup Output and Deployment

Compared to the development of stand-alone mapping systems, the cost of using Google APIs is much less. But comparing to some simple APIs like Yahoo simple APIs, developers need to understand more about AJAX and XHTML in order to use Google APIs properly. Although the Google API Web site supplies complete service of documentation, references, examples, and even templates, to understand all the principles about AJAX is still not that easy as of REST request proceeding.

The deployment and publishing of map mashups created using Google APIs are independent from the server for Google Map (a benefit from the Ajax Technology), so it can be published as any kind of Web application regardless of the server, client or rich-client. Since Google Map APIs use AJAX technology, it can convert static Web pages to dynamic Web pages; in another word, we can deploy rich client applications to client computers without sacrificing the simplicity of Web application. Therefore, AJAX based Google Maps can use some functions from client to reduce the press of the server. At the same time, as the AJAX technology is based on XHTML script language which can

be supported by any browser, the browser does not need install any plug-in for Google Map end-users.

3.3 Yahoo! Map and APIs

3.3.1 Introduction

The Yahoo! Develop Network offers Web Services and APIs that provide services for developers to build applications and Mashups that integrate data sources in new ways, making the web a more useful and fun place for everyone. Yahoo! Maps Simple API makes it a very easy way to display the developers' own geographical content, build a store locator, plan a pub crawl, or create custom routes. To make things even easier, the API can automatically geocode actual address information for the developer — who can no more mess with pre-processing to generate XML that is already encoded with latitude/longitude. At the lowest level, each Yahoo! Maps API works the same way. Using REST, developer can plot points on a map with features like custom descriptions, URLs, labels, groups, icons and more. Data inputs need to be formatted in Web-standard RSS format with either longitude/latitude or address information. More information on Yahoo! Map APIs may be found at <http://developer.yahoo.com/>.

3.3.2 Evaluation

The end-user interface provided by the Yahoo! Maps is almost the same as Google Map on the aspects of map browsing, information navigation and services and pricing. However, Yahoo! Map supports multi-points analysis on the aspect of mapping routes, as for the routes analysis, Yahoo! Map is superior to other map services. Further more,

with the same support of AJAX API, Yahoo! Map API supplies much more options for developers in compatible applications.

For the data updating and accuracy, Yahoo! Map is sometimes unstable for the address matching of area in Canada and sometimes you can not find the information while typing the same address. Compared with Google Maps, the satellite map in Canada is quite old, at least three or four years older; many places can not be found out. Following the evaluation criteria, Yahoo! Map APIs are discussed below.

1. Mashup Data/Service

Yahoo! Map API is a mashup platform that is designed to handle Yahoo! Map data and all related resources and services like searching, locating, routing, ranging etc.

Yahoo Map APIs include the following versions:

Yahoo! Maps Simple API: An easy way to overlay custom map data using the Yahoo! Maps web site to display custom maps. No programming is necessary.

Yahoo! Maps Flash API: Embed maps in custom web site or application using the free Macromedia Flash player for a rich user experience.

- AS-Flash API - Create a Flash application that displays Yahoo! Maps using ActionScript®.
- JS-Flash API - Create great applications quickly with scriptable Flash maps and JavaScript™. No Flash programming is required.

- Flex™ API – Work with Macromedia's Presentation Server Technology to create powerful applications.

Yahoo! Maps AJAX API: Use the power of DHTML and JavaScript to host custom maps.

Map Image API: Programmatically fetching map images with the Geocoding API which can transform physical addresses into encoded latitude-longitude (geocoded) responses. Due to these different kinds of Yahoo APIs, this mashup platform can work with any type of data which are supported by the applications or technology above. For example, based on the powerful support in multimedia data including video, audio etc. by Macromedia Flash, Yahoo! Map API can work with any type of multimedia data through any of the Yahoo! Map Flash API: AS-Flash API, JS-Flash API, or Flex™ API.

2. Mashup APIs and Building Tools

Local files and system wide resources can not be integrated in straightforward manner with this mashup platform considering Internet security. But it can be indirectly integrated with them through a middleware which bridges the Yahoo! Map portable applications and local files or system wide resources. For example, through ASP (Active Server Page) technology, Flash applications can access local files by sending a message to ASP programs running on the local server computer. And then, the Yahoo! Map which is already mashed up in the flash application can also access the local files indirectly through the file management capacity of ASP.

Yahoo! Map APIs do not provide an individual mashup building environment. However, it provides a sort of plug-ins and brief reference documents on its web site. For example Yahoo! Maps MXP (Macromedia® eXtensions Package) is a kind of flash plug-in to support Yahoo Map AS-Flash API in Macromedia Flash 7 Professional or upper version. To use it, just install it through Macromedia Extension Manager, then find it in the component panel inside Flash Professional workspace and drag it out to the work space of flash.

3. Mashup Output and Deployment

Before creating mashup APIs, a developer needs to understand principles each type of API's and how they work. For Yahoo APIs, developers need to understand not only Yahoo APIs but also Flash or XML and some kind of Web Technology like XHTML, DHTML, ASP, PHP, etc. to develop a complete mashup program. But comparing to learning the whole structure and principles of Yahoo! Map and how it works, it becomes an easy job to use just these APIs. Since Yahoo APIs have already encapsulated most of foundation operation code to developers, it makes programming more comprehensible and simplifies the process of program coding.

As mentioned before, with the help of flash, XML or other supported web media platforms, Yahoo! Map APIs can communicate with and mashup with any type of web application including client XHTML, DHTML or CSS and server ASP or PHP. And through XML based REST process, Yahoo! Map API can deploy to a runtime or rich client using language of C# or VB.NET.

Theoretically, Yahoo! Map API can be accessed by any type of browser. It depends on the web application which the Yahoo! Map is based on. For example, in case of a Yahoo! Map mashup with a Flash movie, the browser needs to install the flash player plug-in to support flash movie, but if the communication happens through REST request and in the form of standard XML at a XHTML page, the browser only needs to support XHTML. In addition, when using an application programmed by C# which mashup with the Yahoo! Map, end-user do not need any browser or plug-in to support the program. It just likes a normal rich client application which can run under any operating system it is compiled for.

3.4 MapQuest APIs

3.4.1 Introduction

MapQuest has been around for almost ten years and it is the first company to develop leading Internet mapping program for the ordinary users to get location and direction information. MapQuest also has recently developed an open API and even encouraged developers to take action by creating a developer's challenge contest. Amazon's A9 Map Beta is a notable MapQuest API implementation since it associates street level images with streets on a map to produce a unique mapping experience. MapQuest Advantage API (latest version 5.1) is a free service that allows users or developers to use JavaScript to easily integrate routing, geocoding, and mapping functions into their website. With this feature-rich API, one can create dynamic routing and mapping applications for personal or community use. More information may be found at <http://company.mapquest.com/mqbs/1.html>.

3.4.2 Evaluation

Compared with Google Map APIs, MapQuest has built-in mapping, routing (driving directions) and advanced overlays capability that can provide the most comprehensive and abundant analysis between the two points. Besides it has built-in geocoding capability which could help developers program much convenient Web mapping/GIS applications.

However, with respect to data browsing and displaying, MapQuest cannot provide satellite images; and for the navigation aspect, it does not have the same level of navigation tools provided by Google Map and Yahoo! Map. In addition, MapQuest does not offer services of the highland map photos (e.g. the overview picture from the overhead of the state parliament building) as Google Map does. Moreover, the appearance of the tools and services of MapQuest have gone through only few changes for many years. The following presents a review of MapQuest APIs based on the evaluation criteria.

1. Mashup Data and Services

As similar to Google Map API on most aspects, MapQuest API handles the map browsing, ranging, routing services and stores all map data on its central server for applications and mashups. MapQuest API mashup platform does not support smooth AJAX client and only uses standard XML data type on standard HTML web pages. So, if developers want to work with other type of data, they need to use ASP or PHP to

convert data type to standard XML first, and then MapQuest API can get the data in the same way as Google Map API does.

2. Mashup APIs and Building Tools

MapQuest Map API cannot be directly integrated with local files and system wide resources. But, it can be indirectly integrated with them through some kind of active server page technology like ASP or PHP to access local files or system wide resources. So far, MapQuest does not provide mashup editor program for developers. Since the API is based on JavaScript language, the only way to make programming easier is to use some kind of text editor for JavaScript such as UltraEdit32 or others.

3. Mashup Output and Deployment

Compared to Google Map API and Yahoo! Map API, MapQuest Map API's overhead costs are less, these benefits from its simplicity. For the same reason, the MapQuest Map API's learning curve will come to its peak much faster and more neatly than any other Map APIs.

Because MapQuest Map API can only use JavaScript simple REST for communication with its server, it can only be deployed to a hosted web application directly. If it needs to be deployed to any Web server, some kind of active server page technology like ASP is required. JavaScript is a script language that should be interpretatively executed only in a web page; and any type of runtime or rich client application should be programmed by

a language that can be compiled and run without web browser. As a result, there is no way to deploy it to any kind of runtime or rich client application for MapQuest Map API.

Similarly, Google Map API and Yahoo! Map APIs, MapQuest Map API also does not need any special support than a browser which supports JavaScript. This makes it an easy way to deploy and publish MapQuest Map API based mashups or applications to the Web.

3.5 Other Map Data Sources and APIs

Ask Map and *Windows Live Local* are other two map data sources available. Unfortunately, Ask.com does not supply an open Map API. So, the following discussion is going to focus on Virtual Earth APIs used by Windows Live Local.

Windows Live Local is a new map data source which was published at end of 2005. The most prominent feature of this map source system is its very attractive way of viewing so-called bird's eye views (for example, Figure 3.2), which adds a 3D dimension to the map image.

Figure 3.2 bird's eye view

Like MapQuest Map API, Virtual Earth API is also based on JavaScript. However, due to the support of ASP.Net AJAX service and seamless combination with Microsoft .Net framework, Virtual Earth API allows creation of mashups with any platform in any ways. As a result, Virtual Earth API can not only be developed as easy as MapQuest Map API in simple REST mode, but also be deployed on to any platform in any condition through ASP.Net as Google Map API.

However, it is generally hard to handle and comprehend a new technology such as the Virtual Earth API. Although its grammar and basic implementation method are the simple, REST developers would still face some unexpected difficulties since few matured examples can be found for referencing and learning. In addition, the 3D view requires much more on system hardware support and higher system performance than any other Map API. All of these aspects will increase the overhead costs and learning time for mashup paradigm development.

3.6 Non-map Data APIs

Besides Map APIs mentioned previously, YouTube APIs and Flickr APIs are two mashup platforms that offer multimedia services other than map data services, which are widely used in a variety of web applications or websites.

3.6.1 YouTube and YouTube API

YouTube, published in February 2005, is the most successful online video service supplier with the most powerful strength and extensive influence (see Figure 3.3). YouTube system deals with millions of video clips every day, providing fine quality services of video uploading, distribution communication, exhibition and browsing for millions of users around the world. Through its stable technical support and rich technology, YouTube provides support towards the video content with multi-format. The video display of this website is achieved by Flash technology.

Figure 3.3 The video played on YouTube

The YouTube data API offers a simple and powerful way to access YouTube's content in the form of Google data API ("GData") feeds. Developers' client application can request a list of videos that match specific search criteria, get access to the most popular YouTube videos, or get public information about YouTube users (such as playlists, subscriptions, or contacts). It is based on the Atom 1.0 and RSS 2.0 syndication formats and the Atom Publishing Protocol (see Section 2.2).

3.6.2 Flickr and Flickr API

In the field of image/photo storage, sharing and organization, Flickr (see Figure 3.4) has revolutionary significance. It can not only manage the image conveniently with collaborative processing, but also provide many functions, such as message, mutation

and adding tags on your photos, releasing them on any blog, providing contacts mechanism service, and set the open authority of the photos, etc.

Figure 3.4 Flickr photo displayed with notes

Flickr used CGI with the extension filename “.gne” and MySQL as the core of service, supported by the AJAX technology at the user end to achieve the higher interactivity, which is not missing on the traditional online photo albums. Users can access Flickr online in real time through any web browser that supports JavaScript. Flickr provides its APIs to the public for free in order to permit program designers and developers to develop and/or extend the external functions, and make the content of Flickr cross the gap between websites and software.

3.6.3 Evaluation

1. Mashup Data/Service

Evidently, YouTube API and Flickr API manage multimedia data and some character-string data required by their services, and handle all the services such as searching, uploading, editing, etc. of the media data. Both two APIs supports multi-types request and response format including standard REST, XML-RPC, GData, etc. Particularly, YouTube supports REST and GData format communication method, and Flickr supports REST, XML-RPC, SOAP Request Formats and REST, XML-RPC, SOAP, JSON, PHP Response Formats. So they can work with these formats of data sources smoothly. Since they all support standard REST and XML communication, they can use ASP or PHP to work with any other type of data as other APIs mentioned before do.

2. Mashup APIs and Building Tools

Similar to the other APIs mentioned before, these APIs also cannot directly integrate with local files and system-wide resources considering Internet security. But it can again be indirectly integrated with them through some kind of active server page technology like ASP or PHP to access local files or system-wide resources. In addition, as also mentioned before, Flickr can work straightforwardly with PHP. For other type of mashup, they have to use standard REST and XML document to communication with each other.

So far, both two companies do not provide specific mashup editor program for developers. Since the APIs is based on JavaScript language, the only way to make programming easier is to use some kind of txt editor for JavaScript like UltraEdit32.

3. Mashup Output and Deployment

Since these APIs simply use JavaScript simple REST or some other form of web communication method for communication with its servers, they can only be deployed to a hosted web application directly and, if necessary, deployed to other servers using some kind of active server page technology like ASP. Due to same reasons discussed before, there is no straight way to deploy them to any kind of runtime or rich client application, but each company supplies a sort of development packages to support almost all the languages used in the rich client programming as a bridge. Although this costs a lot of extra time for developers' learning and system communication, this kind of bridge provides convenience for mashup development.

3.7 Summary

MapQuest is the most popular mapping service but lags on features and usability. Google provides the most notable and rapidly emerging web mapping technology, and has a ubiquitous set of APIs. Windows Live Local dazzles with its creative views and features but falls short of the others in direction related functionality. MapQuest offers a number of features but still is missing capability of supporting satellite imagery, which makes it less competitive, compared to its competitors with respect to the core functionality supported. Ask Maps is a worthy competitor but had the highest error rate among those in the group.

Figure 3.5 illustrates sample views of an area displayed using different map data sources from Ask, Windows Live, Google, Yahoo, and MapQuest.

Figure 3.5 Sample views from different map data sources

(Source: [Gruber, 2006])

Table 3.2 presents a summary of a comparison results in terms of map views, navigation, mapping routes, actions, and services and pricing. All map data and services discussed above are included in the table. Apparently, Google and Yahoo provide the most commonly used functions as well as map, satellite, or hybrid view of the data. The only functions supported by all different data services are map view, perform a link, and email map.

Table 3.2 Functionality comparison of different map data services

	Ask Maps	Google Maps	Mapquest	Windows Live Local	Yahoo Maps Beta
Views					
Map View	●	●	●	●	●
Satellite View	●	●		●	●
Map+Satellite View	●	●		●	●
Bird's Eye View				●	
Navigation					
Overview Map Pane		●			●
Draggable Map	●	●		●	●
Shortcut Keys	●	●			●
Mapping Routes					
Driving Directions	Multipoint	A to B	A to B	A to B	Multipoint
Walking Route	●		●	●	●
Reverse Route	●	●	●		
Shortest Time			●		
Shortest Distance			●		
Avoid Tolls			●		
Avoid Highways			●		
Actions					
Permalink	●	●	●	●	●
Email Map	●	●	●	●	●
Send to Phone		●	●	●	●
Live Traffic				●	●
Printable	●	●	●	●	●
Add to Favorites	●				
Buy Prints	●				
Services & Pricing					
Mobile Option		●	●		●
Mobile Pricing		Free	\$3.99		Free
API		●	●	●	●
API Pricing	N/A	Free	\$5K+/year	varies	Free
Notable Uses	N/A	Ning	A9 Maps	Zillow	N/A

*Send to Phone is only enabled for sending business listing from Google Local.

(Source: [Gruber, 2006])

In terms of map APIs, they all support REST and standard XML documents. Google Map API is the most promising one of all the APIs and its full internal AJAX support lets programmers easily mashup without any bridging program or middleware. The Yahoo! Map API has support of the most platforms. This enables programmers to choose any platform they want. MapQuest Map API is the simplest one to learn. Virtual Earth API shows the latest map technology and a new trend of Map APIs' development in the future. Table 3.3 shows the communication protocols supported by each API.

Table 3.3 Comparison of communication protocols supported by APIs

	Language	REST	GData	XML-RPC	AJAX
Google Map API	JavaScript	√	√	√	√
Yahoo Map API	JavaScript ActionScript	√		√	√
MapQuest API	JavaScript	√			
Virtual Earth API	JavaScript VBScript	√		√	
YouTbue API	JavaScript	√	√	√	
Flickr API	JavaScript	√		√	√

Chapter 4 Case Studies: Mashing up Selected PGIS

This chapter presents two cases for studying how the concept of map mashup can be applied to quickly develop some participatory GIS. A prototype for each case study was developed and is used here to facilitate the discussion.

4.1 PPGIS for Municipal Planning Support

By-laws and regulations of many municipalities and local governments require certain level of public participation during municipal planning process (Li et al, 2004). PPGIS, especially web-based, have been widely accepted as an efficient means of supporting public involvement in decision making processes. Historically, PPGIS has been primarily developed using various commercially-available software platforms. Recently, open source solutions have also been adopted in developing different types of PPGIS. With the rapid development of mashup related technologies, mashing up PPGIS may become a viable option. This case study tries to develop a map mashup that allows the public to explore map-based planning information, provide their input, and express their opinions on the proposal plan.

4.1.1 Requirements of Mashup

The Municipal Class Environmental Assessment (MCEA) outlines a procedure whereby municipalities can comply with the requirements of the Environmental Assessment Act. Public participation is an important requirement of MCEA process. As illustrated in Figure 4.1, a number of public meetings and/or public notices are normally carried out to collect public input in a municipal development.

Figure 4.1 Municipal environmental assessment process (after [OGRA, 2000])

Li et al. (2004) summarized the problems associated with the existing practices of collecting public input as: “1) the public is often unprepared when situations requiring participation emerge and they react to circumstances determined by project proponents; 2) interested citizens have to physically attend the public meeting to be able to understand the related issues and voice their concerns and/or comments, which is very often difficult for them given their other commitments; and 3) the existing approach does not provide a rich platform to support interactive public input prior, during, and after public meetings”. The mashup prototyped in this case study aims at facilitating the public understanding of the project under development, participation without physically attending the public meetings, and alternative ways of collecting the public input.

To accomplish these goals, the mashup needs to allow the public to explore the spatial context of the proposed development by providing basic GIS tools such as zoom, pan and identify features. It also needs to provide sketch and annotation tools that allow the participations to mark the maps to add their comments specific to any locations or map features so as to better express their concerns. In addition, explicit general public consensus may need to be collected through voting on the proposed scenarios. A voting tool integrated in such a mashup helps to achieve this. Since the mashup uses data from different sources, tools should be provided to allow municipality department staff or project proponents to define the development area and planned scenarios, and the local achieve of the collected input.

Although not very often seen in current practices, it is believed in some cases that geo-referenced audio or video clips, posted by the public, will be very useful inputs and help explain the participants' concerns. The success of such web portals as YouTube has proved that once the necessary tool is available, the citizens will not be reluctant to use it. As such, the mashup will also include tools allowing the public to post this media type of data.

4.1.2 Mashup Prototype

1. Functional Design Considerations

According to the above general requirements, a user's view of the functional design is illustrated in Figure 4.2. The public participants can view and input on the proposed development by sketching, posting comments and uploading video clips, if they have.

They will also be able to create buffer zones to explore the impact of the proposed development on its surroundings, e.g., if it is “in-my-backyard”. On the other hand, the project proponents or city staff, which are responsible for preparing data, can layout and sketch the plan or alternative scenarios. Certainly, they can view the public input at any time. A server-side agent, software component, is designed to archive all data entered into mashup on a local server other than data services’ provider’s servers.

Figure 4.2 User's view of the functional design

Figure 4.3 shows a typical participation process using the mashup functions. The process captures main paths a user may decide to go through either simply viewing others' input and make his/her own contribution, or performing some basic analysis before contributing. The process illustrated does not capture exceptions such "zero" participation. Together with the user's view of functional requirements, the process provides a guide for the mashup interface design.

Figure 4.3 Participation process containing different paths

2. Mashup Creation and Interfaces

Both Google Map APIs and YouTube APIs are selected in creating the mashup for this case study. The required map data is provided by Google Map, and the videos posted are managed on the YouTube data server. The storage and exchange of the data are achieved by using the techniques combined with Ajax and Microsoft Active Server Page (ASP) technology. JavaScript is mainly used to extend Google Map APIs for map

related operations. In the meantime, the textual data are combined with media such as video by integrating PHP and YouTube. The spatial data (coordinates for posted video and project layout) and the related attribute data (e.g., YouTube address for the posted video) are stored in the form of XML on local server. The non-threaded messages posted can be stored, inquired and counted by using the ASP technique integrated with Microsoft ACCESS databases on the local server to manage messages and provide statistics. Figure 4.4 illustrates the information flowchart showing how the data and messages flow in the mashup.

Figure 4.4 Information and data flowchart

The system interface design is divided into two main parts: Interface for project management and Interface for the public. The project management is an interface for

data entry and management (see Figure 4.5). First, the city staff or project manager needs to define the location and layout of the proposed development (project) using either point or line marker tools. After ensuring the project location and/or layout, the textual data such as names and detailed descriptions, and the videos related to the project can be attached. If applicable, they have option to define buffers surrounding the proposed development to indicate the impact areas.

Figure 4.5 Interface for project management

As seen in Figure 4.5, the panel on the right-hand side has some management functions. The first of them is the “Edit The Project”, which is designed for managing all development projects under public consultation. To be more specific, the project “tree

view” shows the project currently available and could be either clicked to show the exact position of the project in the map or deleted by using the button “Delete Project”. The buffer size is to change the impact range of the project according to pre-defined buffer distance (see Figure 4.6).

Figure 4.6 Project management: buffering

The second set of functions are “Create a New Point” (see Figure 4.7) and “Create a New Line or Area” (see Figure 4.8), which allows city staff or project managers to create a new project no matter it is a point, a line or a polygon. In the project creation area, they can enter the name of the project, upload video (see Figure 4.9), create the project

location by either simply clicking on the map or type the specific coordinates into the textbox, in addition to the buffering scope and the project description.

Figure 4.7 Project management: Create a New Point project

Figure 4.8 Project management: Create a New Line or Area project

Figure 4.9 Uploading video

Interface for the public (see Figure 4.10) is the interface for both city staff/project manager and the public to search information, play videos (see Figure 4.11), analyze information, view feedback, post and check messages (see Figure 4.12). The project layout (point or line markers) will show up on the centre of map after a user chooses a project of interest from project list. If the user clicks on the label, the description for the project would pop up. Also, the related video and messages for the project can be checked by clicking on “show video” and “check message” links. In addition, the related message and opinions from a user can be post by user using the interface shown in Figure 4.13.

Figure 4.10 Interface for the public

Figure 4.11 Playing video related to the project

Figure 4.12 View posted messages related to the project

Figure 4.13 Post message and opinion

3. Mashup Design and Technology Components

At the architecture level, the mashup can be mainly divided into three components according to its functional design considerations, including map operations, message posting and management, and video posting and management. For example, map operations component is composed of viewing of maps and images and analysis of buffering. Among them, the viewing functions provided by Google Map APIs, include typical GIS functions such as pan, zoom and scale, as well as selection and display of the toolbar in the map window. Map operations also allow the annotation of points, lines, polygons and the textual information attached to these annotated features on either map or image views, which can then be displayed to other users. An important feature of the developed mashup is its “buffering” function, which allows users to define a buffer

distance and creates the corresponding buffer around the proposed development such as the central line of a new road.

The message posting and data exchange are the same as the map operation, as shown in Figure 4.14. The message posting component adopts the techniques of ASP and ACCESS database to achieve the storage and reading of the messages posted. Data exchange is mainly divided into data uploading, data storing and data downloading. The data referred include relevant information of map, other message posted and multimedia data etc. Multimedia data handling is achieved by using PHP and APIS provided by the YouTube. Figure 4.14 shows an overall architecture of the mashup developed.

Figure 4.14 Mashup architecture and interactions between components

- Details on the design of data storage structure

The storage of data uses XML-structuring mode. In the data storage, XML is different from database such as Access, Oracle and SQL Server, which provide strong data storage and analysis ability, such as data index, arrangement, search, relevant

consistency etc., while XML only focuses on modeling the data. XML can be read easily in any application program, which means that the program can be easily integrated with the information produced by Windows, Mac OS, Linux and other platforms, and XML data can be easily loaded to the program and analyzed. Finally, the output of results is also in XML form.

According to the resources, “marker” and “line” are designed as two types of nodes to store the resources of place and route, respectively. Every place and route has one “name” attribute. For the place, it is stated by the corresponding latitude and longitude coordinate, and the route, every inflexion point of which is recorded sequentially according to sequence “point” stated by many latitude and longitude coordinate. Meanwhile, there are nodes of “detail description” and “upload video”, respectively to describe the relevant information of this place or route (See Table 4.1).

Table 4.1 Storage structure table for place and route

Marker		
Attribute	Name	
	Latitude	
	Longitude	
	Buffering-Size	
Node	Detail-Description-&-Project video	

Line		
Attribute	Name	
	Buffering-Size	
Node	Detail-Description-&-Project video	
	Point-1	Latitude
		Longitude
		Latitude
		Longitude
	Point-N	Latitude
		Longitude

When the data is being displayed, the mashup program can inquire corresponding information in the XML document according to the place and name of the route. The label can be marked on the map, and the relevant detailed description can be popped up in a bubble window when the client clicks on the label.

• Google MAP API and System's Mashup

Google Maps provides a highly responsive, intuitive mapping interface with detailed street and aerial imagery data embedded. In addition, map controls can be embedded in the product to give users full control over map navigation and the display of street and imagery data. Additionally, users can also perform map panning through the use of the arrow keys on a keyboard as well as dragging the map using the mouse. These

combined capabilities provide a compelling product, but the primary driver behind its rapid acceptance as an Internet mapping viewer is the ability to customize the map to fit into application specific needs. For instance, a real estate agency might develop a web based application that allows end user searching for residential properties, the results of which could be displayed on a Google Maps application. The ability to customize the map display through the addition of application specific data is the true driver of its acceptance as a geospatial viewing tool. Using the APIs, numerous people have created useful and interesting "mashups", combining the Google Maps interface with geographic information from other data centers. (Detail coding operations can be refer to Appendix B)

- **Integrating YouTube APIs**

In order to upload and play videos within the map mashup, YouTube APIs are used to create video uploading component. As discussed before, ASP technique is required to realize the integration of YouTube video on Google maps. Since the communication between ASP and YouTube APIs is not mature and difficult to achieve, another open server PHP is used because it is more mature, stable with better compatibility as a carrier to communicate with YouTube APIs. Figure 4.15 illustrates how the YouTube Video is integrated through ASP and PHP.

Figure 4.15 Integrating YouTube video via ASP and PHP

In fact, the communication between a PHP program attached on the ASP page by the hyperlink and YouTube server makes it possible to realize the function of the video uploading. Then the returned address will be recorded in the standard XML text for the users to load and play the video on their side using the player program provided by YouTube and following the video address recorded in XML format.

To summarize all above, two Internet services, Google Map APIs and YouTube APIs are used to achieve participatory GIS mashup for future e-government services development with mashup technology. The results demonstrated in this paper shows that mashups technology is a helpful tool to develop web-based mapping services in following aspects:

- 1) Rapid development
- 2) Light-weight system architecture
- 3) Friendly user interfaces
- 4) Good expendability

4.2 *West Nile Reporting Services*

As a member of a family of viruses called Flaviviridae, West Nile is normally spread by mosquitoes that have fed on the blood of infected birds according to Public Health Agency of Canada. Reporting dead birds is the first thing the public can do to help identify West Nile virus in our community to stop the spread of this virus. The current reporting mechanism at three levels of governments still uses traditional forms which yet have an important geographical coordinate requirement. This case study tries to develop a map mashup which allows the public to report dead bird with more precise location using a map interface, facilitated by other data entry interfaces.

4.2.1 Requirements for Mashup

The form-based reporting of dead birds requires different types of data depending on the government agencies interacting with. Basically, the required data includes information about the finder of the dead bird, the reporter if different from the finder, the location of the dead bird, and the description of the bird. For the location information, most provinces require the closest address in city and GPS coordinates (latitude and longitude) in rural areas. Appendix A includes such reporting forms used in Alberta and Ontario provinces as examples. These example forms were also used in the case study to decide what data needs to be collected by map mashup.

One of the important elements in the reporting is to identify the bird type, based on the bird pictures and descriptions posted on related government agencies or organization's websites. It is not surprising that reporters often make the incorrect judgement because the dead bird may look out of its shape as shown on the picture. Therefore, allowing the

reporters to attach a photo, if captured, will definitely help the agencies to further verify the bird type. As such, the required data to report the dead birds using the proposed map mashup should include:

- Dead bird finder's information
- Reporter's information, if different from the finder's one
- Dead bird location information, allowing both map pinpointing and direct input of civic address and/or coordinates
- Description of the reported bird and its surround environment

The idea of this case study is to provide a mashup that allows the public to report suspicious dead bird by entering required textual information, uploading pictures and/or videos if any, and identifying the location where the dead birds are found. To accomplish this, the mashup needs APIs for map data, photo and textual data. Ideally, Google APIs that provide APIs for both maps provided by Google Maps and photos provided by Google Picasa should be used because of their technology consistency and compatibility. However, one of the objectives of this thesis project is to study the potential of integrating mashup APIs from different sources. It was therefore decided to use APIs provided by Yahoo! Map and Flickr.

4.2.2 Mashup Prototype

Figure 4.1 Functional Design Considerations

According to the above ideas and principles and to keep it simple, the prototype mashup is designed based on the West Nile Reporting Form in Ontario and Nunavut

Regions. Users can log into the system over the Internet after finding dead birds. Then they can browser the maps, satellite images or Birdseye's view map, and choose and click the location of dead birds through maps. Triggered by the mouse clicks, the mashup will automatically display the clicked location on the map, and then pop up the input page to allow reporters to enter required textual data and upload photos. Finally, all entered and uploaded information will be stored on the server automatically.

Of course, users of the mashup can view the past record of the reported dead birds and relevant information uploaded by other reporters at any time through clicking on the system case tree. Figure 4.16 illustrates process of reporting new dead bird case and browsing the recorded cases using the developed mashup.

Figure 4.16 West Nile reporting and browsing flowchart

Flash satisfies designers' cravings for more graphics and more control over those graphics by providing a way to deliver vector images over the web. One of the advantages of using vector images is to keep file size small. The vector images are also scaleable, which means that one can maintain the control of what a web site looks like when viewers resize the browser window, for example, making the whole thing stay in proportion as the window grows or shrinks. In addition, Flash provides streaming capability. Streaming allows some elements to display immediately upon download while more information continues to arrive over the Internet.

Flash helps beginners create simple animated graphics, but anyone who is familiar with animation can use Flash tools to create quite complex animations with Flash scripting language. ActionScript provides such assisted modes that help beginners to add simple interactive elements. ActionScript is powerful enough to create highly sophisticated interface and application designs using serious scripters. Flash's ability to incorporate sounds and video also make it easy to bring even more live websites. With the advance of flash technologies, Macromedia has added features and functions that expand the program's capabilities as an animation machine and interactivity creator while preserving the easy-to-use drawing tools and assisted animation and interactivity features (Ulrich 2002).

Due to the advantage above, we choose Flash to develop the mashup for West Nile Report system.

2. Mashup Creation and Interfaces

Yahoo! Map and Flickr were selected to create the mashup prototype for this case to study potential integration issues. And as mentioned in Section 4.2, through the use of Yahoo! Map APIs' ActionScript components API, Flash becomes the bridge of the APIs. Through further coding using ActionScript scripting language, Yahoo's maps can be conveniently integrated into flash files. With the powerful multimedia features of flash, the prototyped mashup can be developed more efficiently, and in the meantime, combined with the non-map API: Flickr APIs. This will allow the realization of the required mashup which integrates map, texts, pictures, and diverse forms of expression to achieve its special functions.

This system's data storage requirements are relatively simple, just some textual data and a picture address. Therefore, in order to simplify the design, we only used ASP technology to supplement the deficiency of flash in reading in files directly, and data was directly stored in a text file, thus it is convenient for browsing. The overall interface structure of the mashup is shown in Figure 4.17.

Figure 4.17 Overall interface structures

Figure 4.18 illustrates the main interface of the developed mashup. The interface is divided into two parts. The left panel serves as the map display panel which allows the reporters to browse the maps to locate where the dead bird is found and pinpoint the location on the map. It also serves as the map-based interface to trigger the action for entering information about the dead bird to be reported. The panel on the right-hand side displays a list of past reported dead bird incidents, and allows the users to view more information about any selected incident, including any uploaded photos.

Figure 4.18 Main interface of the West Nile reporting mashup

Figure 4.19 shows a small popup window asking the reporter to confirm the location of the reported dead bird. The interface displays coordinates or civic address information according to the reporter's click location on the map.

Figure 4.19 Dead bird location confirmation interface

After confirming the location, the mashup asks the reporter to specify the type of the bird he/she is reporting by presenting a Type of the Bird interface created using Flash. To facilitate the reporter in identifying the type of the bird, tips in the form of bird photo and brief description are included and can be viewed based on the selection of the bird using radio buttons (see Figure 4.20).

Figure 4.20 Bird type selection interface

Figure 4.22 Mashup architecture showing interactions between components

Yahoo! API's Flash component is embedded into Flash editor in the form of a component package (Yahoomap.mxp). In order to connect it to Flash Professional, it is necessary to open Adobe Extension Manager to add this component, and then to add various support for API to the script of flash using structures similar to import "*com.yahoo.maps.tools.*;*". For the actual realization, a yahoo map is first added to flash and named. A script is then used to add drag and multiple browsing model (see the code in Appendix C for detail).

The realization of Flickr APIs in mashup creation mainly focuses on display. After uploading pictures, the user enters the unique code of a photo obtained from Flickr. And then the mashup accesses the Flickr photo servers through Flickr APIs for the real

address of the photo and then displays the photo on its main interface. The code for realizing this procedure is included in Appendix C.

3. Problems encountered

There have been problems encountered during the development of the above mashup. While some of them have been mentioned previously, the following one is worth further discussion.

The problem encountered during mashup development is the lack of support of Flickr APIs to interact with flash components in the developed mashup. As Flickr API provides only the standard XML communication interface, additional functions are needed to achieve communications in integrating it into the flash. A third party Flashr package, Flashr, is adopted for supporting this kind of communication.

Flashr is an ActionScript 2 wrapper for the Flickr API given by Kelvin Luck (Luck 2007). It makes it easy to develop Flash applications to display and interact with photos uploaded and stored on *flickr.com*. Flashr aims at the ActionScript developer who is familiar with ActionScript 2 and wants to make something utilizing the Flickr API. There is no need to worry about the XML that Flickr returns when one deals with objects like Photos, Photosets and Persons, and about XML parsing, because the code will become very simple and readable (Luck 2007).

Since Flash 7 and its previous version do not support the path to the file operations, the Flashr does not develop corresponding components to support a built-in upload function. Meanwhile, though file operations are supported since Flash 8, according to Macromedia official documents, only path and size information is included, which can not meet the demand of the Flickr API and hence cause compatibility problem. So far, we can only resolve the specific information of the picture and display it using Flickr APIs after it has been uploaded manually through the webpage. The process of uploading photos by the reporter is shown in Figure 4.22.

Figure 4.23 Photo uploading process

4.3 Study Results

The above mashup prototyping effort and study, using various APIs, entailed experience of the convenience and efficiency these APIs brought to users and developers. During this process, a deeper understanding of mashup and its supporting technologies, particularly map APIs, has been established. The following presents a brief summary of the advancement and the limitation concerning map mashup.

4.3.1 Advancements

Participatory GIS shared many characteristics of a large number of mashups, because mashups emphasize end user's involvement and social networking. This makes mashup a great potential for the purpose of developing online participatory GIS applications.

Flexibility: Almost all APIs are independent, meaning there is no need to worry about correlation and dependence between them. Mashup system enables us to place various APIs more appropriately on the most suitable location. Each API plays a comprehensive function without dependence on each other. This is like removable building blocks, which can satisfy various ideas and creativity effectively. What flexibility brought are not only meeting people's various needs well, but also supply a lot of opportunities for software development, even to create some unbelievable effects. When a new technology was given to birth with an API, we can quickly combine it with an existing or another new technology together with flexibility, this proceeding even may soon produce another new technology, and this will surely bring people a lot of imagine space and inspiration.

Easy development and mastery: The development of APIs is for providing interface features; most APIs provided for users are easy to use and master. After some study, majority of developers, or even people who do not have programming and developing experience, can master some simple API usage. Meanwhile, APIs provide some independent functions and services through the software in which the API works. Users can save a lot of development time by directly using existing functions. Developers can also develop public modules from their new needs so as to enhance the efficiency of the development and maintenance with lower cost. For example, when the website for citizen suggestions on municipal planning was designed, some new functions such as the one which can display a 'line' with certain order were packed, by instantiating it from a basic function in the API provided by Google map. Then it can be invoked at any time when such function is needed in the program of the website. Therefore, accumulation and sharing of APIs can greatly enhance the efficiency of the development process; moreover, when doing modification and maintenance, only the common modules with the same function need to be extended.

Furthermore, as the thesis project concerns an audience-oriented GIS website, the website design must comply with people's use habits; in another word it is at least to provide convenient browsing and message posting functions. Mashup allows rapidly adding of function modules which are needed in the website. Moreover, on the map some tools that can update ordinary users' using experience can also be conveniently added.

Full utility of resources: As mentioned above, such resources can bring in greater efficiency and effectiveness. However, the map data sources, such as Google map, Yahoo map, and so on, provide spatial information resources, which are not accessible for ordinary end users or casual developers, and are difficult to be entirely localized. Massive map data and image data can be stored together in a map server provided by these companies. Most people only care about a small portion of the map data, so they can access to the interface of map server created by the API for data, and this is enough to meet all the individual needs. Moreover, the use of these APIs do not need to worry about updating and maintenance of map resources; instead, they only need to browse the map through the Internet any time they want, what they see would be the latest maps and data. In this way, resources would be fully exploited and more reasonably managed.

4.3.2 Limitations

In addition to the flexibility, convenience and efficiency brought about by mashups, there are also some limitations.

API Interoperability: interoperability among different APIs used for creating mashup still are not fully supported, meaning that working with different APIs towards an mashup sometimes are not an easy task, although mashup itself means “integrated”. In this case, extra effort is required to bridge the gaps, as for integrating flash and Flickr technologies in the West Nile case study.

Data Updating and Control: Currently, the data sources used in mashups are all stored, hosted and controlled by the data sources providers. When these data sources

are accessed through their associated APIs, even the user data entered are also posted to the data provider's server. For the mashups used for participatory GIS or government online services, they will be problematic because of the data security, privacy and ownership concerns. The two mashups developed tried ways to keep a copy of the user entered data on the local server other than the provider's server. However, a few problems remain at least not well solved: 1) storing a copy of user entered data on the local server does not prevent the data being sent to the provider's server; and 2) the way the data stored on the local server is not efficient, e.g., missing the map data associated and only the captured coordinates kept.

Limited Interface Expansion: While various APIs provide interfaces for easier master and use, some complicated and cumbersome calculating and processing procedures are packaged into their own program and can not be seen by mashup developers. Therefore, while gaining convenience, the flexible operation of bottom application is sacrificed. Such reasoning itself is the two contradictory aspects in the software development process. Therefore, in dealing with some more complex business, using such packaged APIs with simple interfaces, there will be some problems in basic controlling, accessing and expanding function.

Chapter 5 Concluding Remarks

5.1 Review of the Thesis Objectives

This thesis projects had outlined two objectives at the beginning. The following summarized the work has been done against the two objectives:

Evaluating existing map data sources, services and APIs: the thesis evaluated the data sources, services and APIs provided by Google, Yahoo, MapQuest, YouTube and Flickr based on the 10 criteria listed in Section 3.1. Special attention was paid to those related to map data and APIs that can be used to create map mashups. The evaluation has found that:

1. Not all map APIs provide geocoding services, for example Google Map APIs. In order to have geocoding function working with Google Map based mashups, a “geocoder” that can be accessed via HTTP request or directly from within the JavaScript is needed. On the other hand, Yahoo! Map API and MapQuest APIs support geocoding capability.
2. All map data sources, services and APIs do not support the uploading data to a local server other than the provider's server. This poses a big limitation on using mashups for those applications such as participatory GIS and online government services, which have a strict concern about privacy and data security and ownership.
3. The map data coverage of the existing data services from Google, Yahoo and MapQuest depend on the areas in question. When the coverage is not complete, or more data coverage (more data layers) is required, the existing map APIs do not

support map data integration, i.e., it is not possible to mashup your local map data with the data from the map data providers by directly overlaying them on each other.

Prototyping map mashups for selected case study: two map mashups have been created using data sources, services and APIs from Google, Yahoo, YouTube and Flickr to study potentials issues related to the use of mashup technologies for developing participatory GIS applications. Special attention was given to the data integration and management, effort in mashup creation, and technology integration. Social and organizational issues related to data ownership and privacy was discussed to a limited extent. Some problems encountered during the two map mashup creation include:

1. The integration of flash technology: when mashup creation involves data sources and APIs from multiple providers, technology incapability and data integration issues are inevitable. As discussed in section 4.2.2.3, an example of this is related to the integration of flash technology and Flickr API.
2. Enormous/voluminous data management issues using XML/TXT: since XML/TXT is widely used in various APIs used in creating mashups, with the increasing amount of data exchanged and stored between mashup components, the management of data in XML/TXT format would become more and more difficult.
3. Data privacy and ownership: one of the major concerns for e-government and public participation online services is the data privacy and ownership. In developing the two mashups, measures were taken to provide alternative solutions for keeping the data as much as possible on a local server and sending minimum information to the

provider's server. However, due to the APIs restrictions, it is not possible to keep both attribute data (text, video, photo, etc.) together with map data on the local server except for the necessary coordinates extracted from the map data. Keeping public input completely private and away from external data providers' servers remains an issue.

5.2 Observations and Recommendations

1. Social problems faced by Mashup

As the mashup becomes more popular, some social problems will arise. The most serious social problem that mashup developers need to face is to maintain the balance between the protection of intellectual property rights, consumer's privacy and the publicity and free flow of information. Mashing up web applications is still in its infancy. There are only some developers writing programs in their spare time, and these developers might not be aware of problems such as security. Moreover, many people do not regard this as the focus of core business, which leads to the current low-quality software. To promote the maturity of the software development process, the whole community must work collaboratively as a group, to set up open standards and reusable toolkits. Before the mashup becomes mature and can be applied to more core-based business and e-government services applications rather than social networking "toys", a lot of work still needs to be done, so as to form a highly robust standards, protocols, models and kits.

2. Mashup facing technical problems

Meanwhile, just like other data integration fields, mashup development is also filled with many technical challenges to be resolved. As the enrichment of mashup features and application functions, this challenge will become even more severe.

Among them, a very important question is about the quality of semantics and data on integration. The survey on quality shows that the primary concern of today's enterprise IT is data integration in virtual organizations of the enterprise. Mashup developers need to face the challenges similar to semantic share of date derived from heterogeneous data sets. Mashup developers may also face a number of problems that IT integration managers do not need to face, for instance, data corruption. As part of the design of applications, many mashups require public users to input. This is a double-edged sword: it can be very powerful, because they can provide an open contribution and optimal data updating. However, it will also lead to inconsistent, incorrect or misleading data items. The latter might endanger the credibility of the data, and ultimately reduce the value, trust and reputation brought by mashups.

In addition, there are challenges from components. Though compared with the traditional whole page refresh technology, the Ajax model developed by Web provides richer and more seamless user experience; however, it also brings some difficulties. Fundamentally, Ajax requests combines the client browser script function with its DOM to achieve a content delivery method, which is entirely envisaged by browser designers. However, this leads to the common browser compatibility problems of Ajax-based

application programs, which have been puzzling Web developers since the development of Internet Explorer by Microsoft.

3. Mashup facing the security issues

Just like any areas of cross-distribution application programs, mashup developers and content providers also need to address some security issues. The concept of identity may become a thorny subject, while the traditional Web is primarily constructed for the anonymous visit. Although, the SSO is a satisfactory property, there exists a variety of competition technologies. Therefore, it could lead to disordered space of identity naming. Vendors may use identity verification and authorization model in their own API for the mandatory use of business model concerning paid subscription or sensitive data. Sensitive data may also require certain confidentiality, which requires that developers must make it clear when to integrate them with other resources set without bringing risks. In addition, since the data integration happens simultaneously at the server and client end, the identity and certificate consignment to mashup services from the user may also be a demand.

References

- Anderson, P. (2007) "What is Web 2.0? Ideas, technologies and implications for education"
- Community GIS Spatial information management, "<http://www.mapunity.in/cgis/>" (last accessed on October 8, 2007)
- Erle, S. R., Gibons & J. Walsh (2005) Mapping Hacks: "Tips & Tools for Electronic" Cartography, O'Reilly Press
- Ennals, R. & Gay, D. (2007) "User-Friendly Functional Programming for Web Mashups", Intel Research Berkeley, Berkeley
- Gruber, F. (2006) "Comparing the Mapping Services", "<http://www.techcrunch.com/2006/04/17/comparing-the-mapping-services/>"
- Google Maps API Concepts, "<http://www.google.com/apis/maps/documentation/index.html>" (last accessed on September 20, 2007)
- Garrett, J.J. (2005) "Ajax: A New Approach to Web Applications", adaptive path Blog, February 18, 2005, "<http://www.adaptivepath.com/ideas/essays/archives/000385.php>"
- Li, S., Y. Ru & Z. Zhang (2004) "Enhancing Online Public Notices Using GIS to Facilitate Public Participation in Municipal Developments", Proceedings of the XXth ISPRS Congress, Istanbul, Turkey, July 12-23, 2004
- Merrill, D. (2006) "Mashups: The new breed of Web app: An introduction to mashups", IBM DeveloperWorks, "<http://www.ibm.com/developerworks/xml/library/x-mashups.html>" (last accessed on October 8, 2007)
- Mapquest (2007) "<http://www.mapquest.com/>" (last accessed on October 7, 2007)
- OGRA, (2000) Municipal Class Environment Assessment, a document obtained from Ontario Good Roads Association
- O'Reilly, T. (2005) "What Is Web 2.0: Design Patterns and Business Models for the Next Generation of Software", an article published on O'Reilly Media, Inc. website, September 30, 2005, <http://www.oreilly.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html>
- Pietroniro, E. & Fichter, D. (2007) "Map Mashups and the Rise of Amateur Cartographers and Mapmakers", ACMLA Bulletin, No. 127, 2007, pp. 26-30

Rubio, D. (2007) "Create a Map Mashup with Oracle Spatial", an article published by Oracle Technology Network, "<http://www.oracle.com/technology/pub/articles/rubio-mashup.html>" published on March 2007

Ruan, H. (2003) "The theoretical research on public participatory GIS and the application in city planning", Fujian: Fuzhou University, 2003

Roy, F. (2002) "Representation State Transfer", <http://www.xfront.com/sld001.htm> (last accessed on October 24, 2007)

Ramsey, P. (2006) "Mashing Up the Enterprise", Geospatial Solutions (online magazine), April, 2006

Ulrich, K. (2002) "Macromedia Flash for Windows & Macintosh", "http://books.google.ca/books?id=AzNfZnOo6SAC&printsec=frontcover&dq=Macromedia+Flash+for+Windows+%26+Macintosh&hl=zh-CN&sig=Q15ns7RTkr6_du7-wW0R1UUjNwA" (last accessed on October 20, 2007)

Wikipedia (2007) [http://en.wikipedia.org/wiki/Mashup_\(web_application_hybrid\)](http://en.wikipedia.org/wiki/Mashup_(web_application_hybrid)) (last accessed on September 10, 2007)

Appendix

The appendix section includes:

- Appendix A: Sample West Nile Virus Surveillance Forms
- Appendix B: Code Segments for study case one
- Appendix C: Code Segments for study case two

Appendix A: Sample West Nile Virus Surveillance Forms

WEST NILE VIRUS SURVEILLANCE FORM – 2005
Alberta Fish and Wildlife

Use this form to accompany each specimen, [corvids only (primarily crows, magpies, blue jays, ravens)] submitted for the West Nile Virus Surveillance Program.

DATE FOUND: _____, 2005

DATE COLLECTED: _____, 2005

LOCATION WHERE BIRD WAS FOUND:

Urban Centres:	Rural Areas:
Street Address:	Legal Land Description: (Qtr - Sec - Twp - Rge - Mer)
City/Town	Directions Distance from Nearest Town
Postal Code (important!): _____	GPS coordinates
	Municipality (R.M.#):

FINDER Name: Street Address: City/Town: Telephone:	SUBMITTER (Usually the Fish and Wildlife office). Name: Agency: Street Address: City/Town: Email: Telephone: FAX:
---	---

Details of Incident and Other Information:

- Was the bird euthanized? (circle) Yes No (if yes, how?) _____
- Carcass stored: (circle) Fresh or Frozen
- Date received: _____
- Other information (please use back of form if necessary):

You must COMPLETE and ATTACH a form with every carcass submitted for West Nile Virus Surveillance.

ONTARIO/NUNAVUT REGION
CANADIAN COOPERATIVE WILDLIFE HEALTH CENTRE

Department of Pathobiology, Ontario Veterinary College
University of Guelph, Guelph, Ontario N1G 2W1
Telephone: Toll free 1-866-673-4781 (ONT) / [519] 824-4120 Ext. 54662
FAX: [519] 821-7520 / 824-5930 / email: ccwhc@uoguelph.ca

INCIDENT REPORT FORM

(to report wildlife disease/mortality incident, and to accompany any specimen submissions)

Date Received	Necropsy No.	CCWHC No.
Name of Submitter Address Postal Code email address Phone # [] [] FAX # [] [] REFERENCE # (if applicable)	Copy Report to Address Postal Code email address: Phone # [] [] FAX # [] []	

Signature of submitter _____ signifies permission to collect the personal data on this form for purposes of reporting, follow-up investigation and geolocating incident. Information is held in a secure database managed according to applicable federal and provincial privacy legislation.

Found by (name, mailing address including Postal Code, telephone)

Locality of Incident:
(urban street address, Rural township, lot, concession, etc. if possible)

Latitude _____ °, _____ deg N, Longitude: _____ °, _____ deg W (use degree decimal WGS84 setting & 4 decimal places)

Species _____
(separate form for each species, please)

Age: _____ Sex: _____

Total number of this species: Appearing Normal _____ Sick _____ Dead _____

DATES (yyyy-mm-dd): Incident occurred: _____ Incident observed: _____ Picked up: _____

Environmental Circumstances surrounding incident/mortality, in full
(eg. weather, proximity to buildings, power lines, use of chemicals, human interaction, etc. - use back of form if necessary)

Behavior/Clinical Signs of Affected Animal(s): _____

Suspected cause of death? _____

Specimens (circle appropriate) No To follow (how/when) _____
Yes Incident previously reported to us? If so, when? _____
Indicate # of each submitted # _____ Carcasses / Parts: Fresh Frozen Fixed
_____ Other (specify) _____

Was specimen Found dead? Found live/died? Killed? (circle) How killed? _____

CCWHC Form 001 (2011)

West Nile Virus Bird Submission Form

This form **MUST** accompany **EACH** specimen (of the crow family only) submitted for WNV testing.

Note: Please do not submit specimens in poor condition (i.e. grossly decomposed carcasses).
Bird specimen submissions with incomplete information or missing forms will **NOT** be tested.

Date Found/Observed: _____, 200__ Species of Bird: (circle)
Date Specimen Collected: _____, 200__ *Crow Raven Magpie*
Carcass Stored: (circle) *Fresh or Frozen* *Steller's Jay Blue Jay*

Location where bird was found (please complete either urban **OR** rural location information as appropriate):

Urban Centres	Rural Areas
Street Address:	Directions/Distance from Nearest Town: e.g. 40 km W of Kamloops on HWY 1/97
City/Town:	GPS Coordinates (Lat-Long in Deg/Min/Sec, NAD83): e.g. N50°46'12" W120°54'00"
Postal Code:	

Local Health Area : _____ (Note: **mandatory** – testing results will be reported by LHA)

Finder (person that identified the bird):

Name:
Telephone:
Email:

Submitter (officer sending the bird for testing):

Name:
Agency:
City/Town:
Telephone:
Email:

Please use the back of this form for details of incident and/or other information.

Specimen submissions to:

Animal Health Centre
Corvid Surveillance Program
1767 Angus Campbell Road
Abbotsford V3G 2M3
Toll free in BC: 1-800-661-9903
Phone: 604-556-3003

Surveillance inquiries to:

BC Centre for Disease Control
Epidemiology Services
Vancouver
Phone: 604-660-6061
Internet: www.bccdc.org

Each completed form should be enclosed in its own waterproof (e.g. Ziplock®) bag and attached to the corresponding bird carcass.

Animal Health Centre use only

Date received by AHC: _____, 200__

AHC Specimen ID: _____

FORM - 1

West Nile Virus - Surveillance

West Nile Virus Hotline (Health Links): 1-888-315-9257 or in Winnipeg 788-8200

Veterinary Services Branch (Tel: 945-8838)

Use this form to report sightings of dead/sick crows, ravens, magpies, blue jays and gray jays
and to accompany specimens submitted for examination for West Nile Virus

Date Sick or Dead Birds were Found/Observed: _____, 2002

Date Specimen(s) Collected: _____, 2002

Has the bird(s) been reported to Health Links? Y / N Health Links Sighting ID Number (Use Health Links number if one has been given - if not use agency number, if no agency number leave blank)	Specimen Collected and Submitted for Testing? (Check one) Yes _____ No _____ Unknown _____	Species- (Indicate how many) Crows _____ Ravens _____ Magpies _____ Blue Jays _____ Gray Jays (Whisky Jacks) _____
Name of Finder: Address: Postal Code: Email: Telephone: FAX:	Name of Submitter (if different than Finder): Address: Postal Code: Email: Telephone: FAX:	

Location where Bird was Found (Describe as precisely as possible):

Street Address:		Land Location, QTR-SEC-TWP-RGE-MER and/or	
City/Town:			
Postal Code:		Directions and Distance from Nearest Town or Distinguishing Landmark	

Species of Birds Found	Number Sick and/or Dead	# Submitted for Examination (if any)

Details of Incident and Other Information: Please use the back of this form

Please fax this form to CCWHC (306) 966-7439

If specimen available submit the form with the dead bird(s) to
Veterinary Services Branch Laboratory
545 University Crescent, Winnipeg MB R3T 5S6

Note: Health Links will fax the form if a Health Links sighting number has been given. A form should accompany all bird submissions.

A2007-
WNV-A2007-

WEST NILE VIRUS SURVEILLANCE FORM – 2007

Atlantic Region Canadian Cooperative Wildlife Health Centre
Department of Pathology and Microbiology AVC, University of Prince Edward Island
Charlottetown, PEI C1A 4P3
Tel: (902) 628-4314 or (902) 566-0667 FAX: (902) 566-0851 Email: djones@upci.ca or daoust@upci.ca

Use this form to accompany each specimen, (crow family only), submitted for the West Nile Virus Surveillance Program.

SPECIES OF BIRD: (circle)

DATE FOUND: _____, 2007

Crow

Raven

Gray Jay

DATE COLLECTED: _____, 2007

Magpie

Blue Jay

LOCATION WHERE BIRD WAS FOUND:

Urban Centres:	Rural Areas:
Street Address:	Legal Land Description: (Qtr Sec Twp Rge Mer)
City/Town:	Directions Distance from Nearest Town:
Postal Code (important!): _____	GPS coordinates:
	Municipality (R.M.#):

FINDER (contact information essential for reporting test results back to you)

Name:

Address:

Telephone:

SUBMITTER (Conservation Officer or person dropping off bird):

Name: _____

Agency: _____

Street Address: _____

City/Town: _____

Email: _____

Telephone: _____

FAX: _____

Details of Incident and Other Information:

- Was the bird euthanized? (circle) Yes No (if yes, how?) _____
- Carcass stored: (circle) Fresh or Frozen
- Date received by CCWHC: _____
- Other information (please use back of form if necessary):

You must COMPLETE and ATTACH a form with every carcass submitted for West Nile Virus Surveillance. Please call (902) 628-4314 for assistance.

W2005-
WNV-W2005-

WEST NILE VIRUS SURVEILLANCE FORM – 2005

Western & Northern Region Canadian Cooperative Wildlife Health Centre
Department of Veterinary Pathology, WCVN, University of Saskatchewan,
52 Campus Drive, Saskatoon S7N 5B4
Tel: (306) 966-1873 Toll Free: 1-866-544-4744 FAX: (306) 966-7439 Email: ccwhc@usask.ca

Use this form to accompany each specimen, (crow family only), submitted for the West Nile Virus Surveillance Program.

DATE FOUND: _____, 2005 Crow Raven Gray Jay
DATE COLLECTED: _____, 2005 Magpie Blue Jay

LOCATION WHERE BIRD WAS FOUND:	
Urban Centres: Street Address: City/Town: Postal Code (important!): _____	Rural Areas: Legal Land Description: (Qtr Sec Twp Rge Mer) Directions Distance from Nearest Town: GPS coordinates: Municipality (R.M.#):

FINDER (contact information essential for reporting test results back to you)

Name:

Address:

Telephone:

PEST CONTROL OFFICER that picked up the bird carcass (if applicable) Name: Street Address: City/Town: Email: Telephone: FAX:	SUBMITTER (Conservation Officer or person dropping off bird): Name: Agency: Street Address: City/Town: Email: Telephone: FAX:
---	---

Details of Incident and Other Information:

- Was the bird euthanized? (circle) Yes No (if yes, how?) _____
- Carcass stored: (circle) Fresh or Frozen
- Date received by CCWHC: _____
- Other information (please use back of form if necessary): _____

You must COMPLETE and ATTACH a form with every carcass submitted for West Nile Virus Surveillance. Please call 1-866-544-4744 for assistance.

Appendix B: Code Segments for study case one

```
function ShowAllPoints()
{
    var points = doc.getElementsByTagName("marker");
    for(var i=0;i<points.length;i++)
    {
        var detail = GetDetailByNode(points[i]);
        ShowPoint( points[i].attributes[2].value,points[i].attributes[1].value,detail);
    }
    map.setCenter(new GLatLng(43.65636,-79.38096), 12);
}

function createMarker(point, detail, icon)
{
    var marker = new GMarker(point);

    if(icon!=null) marker.icon = icon;

    GEvent.addListener(marker, "click", function()
    {
        marker.openInfoWindowHtml("Description: <b>" + detail + "</b>");
    });
    return marker;
}

function GetDetailByNode(xmlnode)
{
    var nodes = xmlnode.getElementsByTagName("detail");
    return nodes[0].text;
}

function ShowPoint(x,y,detail)
{
    var point = new GLatLng(x,y);
    map.addOverlay(createMarker(point, detail));
}

function ShowStart(x,y,detail)
{
    var icon = new GIcon();
    icon.image = "/images/start.png";
    icon.iconSize = new GSize(20, 30);
    icon.iconAnchor = new GPoint(18,30);
    icon.infoWindowAnchor = new GPoint(5, 1);
}
```

```

        var point = new GLatLng(x,y);
        //map.addOverlay(new GMarker(point, icon));
        map.addOverlay(createMarker(point, detail,icon),icon);
    }
    function ShowEnd(x,y,detail)
    {
        var icon = new GIcon();
        icon.image = "./images/end.png";
        icon.iconSize = new GSize(20, 30);
        icon.iconAnchor = new GPoint(18,30);
        icon.infoWindowAnchor = new GPoint(5, 1);
        var point = new GLatLng(x,y);
        map.addOverlay(createMarker(point, detail,icon),icon);
    }

    function ShowAllLines()
    {
        var lines = doc.getElementsByTagName("line");

        for(var i=0;i<lines.length;i++)
        {
            var points = lines[i].getElementsByTagName("point");

            var lastpoint = new
            GPoint(points[0].attributes[0].value,points[0].attributes[1].value);

            ShowStart(points[0].attributes[1].value,points[0].attributes[0].value,GetDe
            tailByNode(lines[i]));
            for(var j=1;j<points.length;j++)
            {
                var thispoint = new
                GPoint(points[j].attributes[0].value,points[j].attributes[1].value);
                ShowLine(lastpoint,thispoint);
                lastpoint = thispoint;
            }
            ShowEnd(points[points.length-
            1].attributes[1].value,points[points.length-
            1].attributes[0].value,GetDetailByNode(lines[i]));
        }
    }

    function ShowLine(p1,p2)
    {
        var line = new GPolyline([p1, p2], "#000000", 5);
        map.addOverlay(line);
        //alert(line);
    }

```



```

    }

function ShowLineInSize(p1,p2,scope)
{
    var lineWidth = parseInt(scope*2 + 2);
    var line_buf = new GPolyline([p1, p2], "#FF0000", lineWidth);
    var line_self = new GPolyline([p1, p2], "#000000", 5);

    map.addOverlay(line_buf);
    map.addOverlay(line_self);
}

function ClearAll()
{
    map.clearOverlays();
}

```

Appendix C: Code Segments for study case two

Add event listeners:

```
myMap.addListener(com.yahoo.maps.api.flash.YahooMap.EVENT_INITIALIZE,  
onInitMap); // initialization of map
```

```
myMap.addListener(com.yahoo.maps.api.flash.YahooMap.EVENT_ONCLICK,  
onClick);
```

```
// response to click event
```

Concrete set up of initialization condition:

```
function onInitMap(eventData) {  
    var panTool = new PanTool();  
    myMap.addTool(panTool, true);           //to Enable Drag  
    var navWidget = new NavigatorWidget("close");  
    myMap.addWidget(navWidget);           //add a mini map  
    myMap.addWidget(new SatelliteControlWidget()); //add a switcher of map view  
}
```

Record the longitude and latitude clicked:

```
function onClick(eventData) {  
    myMap.removeAllMarkers();  
    var myMarker:Object = {index:"Here !", title:'Something Special', description:'Found  
something Here!', markerColor:0x990099, strokeColor:0xFFFF00};  
    myMap.addMarkerByLatLon(CustomPOIMarker, eventData.latlon, myMarker);  
    _root.gotoAndPlay(2);  
    CurLatlon=eventData.latlon;
```

```

var lat=CurLatlon.lat.toString();
var lon=CurLatlon.lon.toString();
CurAddress="Lat:"+lat.substr(0,5)+"Lon:"+lon.substr(0,5);
}

import com.dynamicflash.utils.Delegate;
import com.kelvinluck.flashr.core.*;

    var P_URL="";           //Real URL of the picture
    var _flashr:Flashr = Flashr.getFlashr();      //Flashr object instance
    var _flashrResponse:FlashrResponse=new FlashrResponse();//FlashrResponse
    // object instance
    _flashr.apiKey = "b40e05adf210ad4c4cc4da00f99f4184";// Flickr API license
    _flashr.cacheQueries = true;
    //Corresponding event listener
    _flashrResponse.onPhotosGetInfo = Delegate.create(this, onPhotosGetInfo);
    var thePhoto:Photo=Photo.getPhoto(theForm.PicURL.text);//Initialize picture
object
    _flashr.photosGetInfo(thePhoto.id);      //Start query of picture
    function onPhotosGetInfo(photo:Photo){
    P_URL+=photo.thumbnailUrl.toString();    //receive the URL and save
    CurListNode=new ListNode(
        theForm.RDate.text,
        theForm.RName.text,
        theForm.RAddress.text,

```

```
theForm.RPostCode.text,  
theForm.REmail.text,  
theForm.RTel.text,  
CurLatlon,  
theForm.IAddress2.text,  
theForm.SType.text,  
theForm.SNumber.text,  
theForm.radioGroup.selectedData,  
theForm.HDate.text,  
theForm.FDate.text,  
theForm.GDate.text,  
theForm.EnvDetail.text,  
P_URL);  
CurList.push(CurListNode);
```

.....