Master's Degree in Informatics Engineering
Internship
Final Report

Crossroads:
Real-time classification of roads based on the city data

Mário Gustavo Reis Caseiro e Alves Pereira
mgreis@student.dei.uc.pt

Supervisors:
Mário Alberto Costa Zenha Rela, PhD
Ricardo Jorge Fernandes Vitorino, MSc
Date: 1st July 2017

Abstract

Humanity is experiencing the largest urban growth in history. Nowadays more than half the human population lives in cities. This rapid growth of urban areas poses great challenges to governments in terms of sustainability, mobility and air quality.

With the proliferation of inexpensive everyday objects, embedded with electronics and able to connect themselves to a network, it has become possible to use them to collect and exchange data.

Using a vast network of sensors deployed over a large metropolitan network, it is possible to autonomously collect data and transmit it to a central system that processes it into valuable information that can assist in city governance and improve the citizen's life.

Crossroads aims at studying the viability of using traffic and air quality data, collected by a large sensor network, to introduce modifications into a Geographic Information System used by a Routing Service that is able to calculate the best route between a source and a target location. Its main objective is to study algorithms, services, and tools that can be applied to build a small prototype that demonstrates the concept.

This report presents and discusses all taken steps and activities developed during the Crossroads internship.

Key words: "Air Quality", "Mobility", "Routing", "Smart Cities", "Traffic", "Web Maps"

Table of Contents

Tables List xi

Figures List xiii

1 Introduction 1

1.1 Motivation 1

1.2 Objectives 1

1.3 Ubiwhere 2

1.4 Political and Social Context 3

1.4.1 Smart Cities 3

1.4.2 Mobility and Transportation European Commission Policies 3

1.4.3 European Innovation Partnership on Smart Cities and Communities 4

1.5 Document Structure 5

1.6 Conclusions 5

2 Background Knowledge 7

2.1 Introduction 7

2.2 The Infrastructure 7

2.2.1 Sensors 7

2.2.2 Real-Time vs Non- Real- Time Sensing System 8

2.2.3 BlipTrack Aarhus Case Study 8

2.2.4 Citibrain Platform case study 9

2.3 Map Sources 11

2.3.1 OpenStreetMap 11

2.3.2 Google Maps 14

2.3.3 Choosing a Map Source 15

2.4 Standards: The Open Geospatial Consortium 15

2.4.1 The Web Map Service 15

2.4.2 The Web Map Tile Service 16
2.5 The Traditional Web Map Routing Implementations 16
2.6 Web Map or Tile Services 17
2.6.1 Mapnik Map Rendering Software tool 17
2.6.2 Deploying our own OpenStreetMap Server 18
2.6.3 Outsourcing the Web Map Service 18
2.6.4 MapProxy 18
2.7 Routing Services 19
2.7.1 Open Source Routing Machine 19
2.7.2 GraphHopper Routing Engine 20
2.8 Geocoding Services 21
2.8.1 Nominatim 21
2.8.2 Outsourcing the Geocoding Service 22
2.9 Conclusions 22

3 Planning and Development Methodologies 23
3.1 Introduction 23
3.2 Methodology 23
3.3 Tools 24
3.3.1 Redmine 24
3.3.2 GitLab 25
3.4 First Semester Planning 25
3.5 Second Semester Planning 26
3.6 Gantt Chart 28
3.7 Conclusions 30

4 Preliminary Activities 31
4.1 Introduction 31
4.2 Preliminary Work 31
4.2.1 Introduction 31
4.2.2 Choosing Routing Web Services 31
4.2.3 Data Sources 32
4.2.4 Web Map Routing Implementation 33
4.2.5 Manipulating Map Information 36
4.2.6 Conclusions 36

4.3 Requirements . . . 37
4.3.1 Functional Requirements . . . 38
4.3.2 Constraints . . . 43
4.3.3 Non-Functional Requirements . . . 43
4.4 Initial High Level Architecture . . . 44
4.4.1 Architectural Drivers . . . 44
4.4.2 Architecturally Significant Requirements . . . 44
4.4.3 Architectural Style . . . 45
4.4.4 System Decomposition . . . 46
4.4.5 System High Level Architecture . . . 47
4.4.6 Conclusions . . . 50
4.5 Technologies . . . 50
4.5.1 Yet Another Django Project Template . . . 51
4.5.2 Programming Language . . . 52
4.5.3 Django Web Framework . . . 52
4.5.4 Message Broker . . . 53
4.5.5 Distributed Task Queue . . . 54
4.5.6 Relational Databases . . . 54
4.5.7 Automatic Deployment Tools . . . 55
4.5.8 Web Server Gateway Interface . . . 56
4.5.9 Web Servers . . . 56
4.5.10 Let's Encrypt and Certbot . . . 56
4.5.11 Libraries for Web Mapping Applications . . . 57
4.6 Conclusions . . . 57

5 Development **59**
5.1 Introduction . . . 59
5.2 Iteration One . . . 60
5.2.1 Planning and Risk Assessment . . . 60
5.2.2 Risk Mitigation Activities . . . 60
5.2.3 Conclusions . . . 62
5.3 Iteration Two . . . 62
5.3.1 Planning and Risk Assessment . . . 62
5.3.2 Convert an array of geographic coordinates into an array of Nodes . 63
5.3.3 Conduct Web Routing experiments using the OSRM Traffic Feature 64

5.3.4 Web Routing Experiment 1 - Unmodified Vs Modified Map Export 66
5.3.5 Experiment 2 - Modified Map Exports: Light vs Moderate vs Heavy Traffic . . . 67
5.3.6 Conclusions . . . 68
5.4 Iteration Three . . . 68
5.4.1 Planning and Risk Assessment . . . 68
5.4.2 Layer the Node Arrays, obtained from the OSM service, on the browser-based web map . . . 69
5.4.3 Determine if Nodes obtained from the OSM service were present on the OSRM Map . . . 71
5.4.4 Evaluate the resulting prototype against GraphHopper traffic data integration demonstration . . . 73
5.4.5 Conclusions . . . 76
5.5 Iteration Four . . . 77
5.5.1 Planning and Risk Assessment . . . 77
5.5.2 Engineering and Construction . . . 77
5.5.3 Conclusions . . . 82
5.6 Testing . . . 82
5.6.1 Introduction . . . 82
5.6.2 Unit Testing . . . 83
5.6.3 Integration Testing . . . 84
5.6.4 Deployment Testing . . . 84
5.6.5 Usability Testing . . . 84
5.6.6 Acceptance Testing . . . 86
5.7 Conclusions . . . 87

6 Results and Conclusions 89
6.1 Introduction . . . 89
6.2 Results . . . 89
6.2.1 Conducted Activities . . . 89
6.2.2 The Browser-based Client Application . . . 91
6.2.3 Project Evaluation . . . 92
6.3 Conclusions . . . 93

A Shortest Path Problem 103
A.1 Map Representation . . . 103

A.2 Linear Programming Solution 104
A.3 Dijkstra's Algorithm 104
A.4 Bidirectional Dijkstra Algorithm 105
A.5 Goal Orientated Search (A*) 107
A.6 Hierarchical Methods 108
A.7 Node and Edge Labeling 109
A.8 Combining techniques 111

B User Stories 113
B.1 Players/stakeholders 113
B.2 Work Division Structure 114
B.3 User Stories Structure 114
B.4 User Stories Definition 115

C Quality Requirements Scenarios - Utility Tree 127

D Initial Architecture 133
D.1 System Decomposition 134
D.2 Data Pipe Flow 135
D.3 Business Processes 136
D.3.1 Database Updater System 136
D.3.2 Compile and Update Routing Engine System 137
D.3.3 Sensor Endpoint 138
D.3.4 Validate 139
D.3.5 Validate Message 140
D.3.6 Analyse Attribute 141
D.3.7 Compile OSRM File 142
D.3.8 Prepare OSRM Map File 143
D.3.9 Make Files Available 144
D.3.10 Load New Map File to Router 145
D.3.11 System Architecture - Layer View 146
D.3.12 System Architecture - Component Diagram 147
D.3.13 System Architecture - Components Architecture Diagram 1 148
D.3.14 System Architecture - Components Architecture Diagram 2 149

E Risk Analysis 151

E.1 Purpose . . . 151
E.1.1 Threshold of Success . . . 151
E.1.2 Risk Identification . . . 152
E.2 Iteration One . . . 153
E.2.1 Risk Identification . . . 153
E.2.2 Risk Prioritization . . . 156
E.2.3 Risk Mitigation Plan . . . 156
E.3 Iteration 2 . . . 157
E.3.1 Overall Risk Evolution . . . 157
E.3.2 Threshold of Success . . . 157
E.3.3 Existing Risks Evolution . . . 157
E.3.4 New Risks Identification . . . 158
E.3.5 Risk Prioritization . . . 159
E.3.6 Risk Mitigation Plan . . . 159
E.4 Iteration 3 . . . 160
E.4.1 Risk Evolution . . . 160
E.4.2 Threshold of Success . . . 160
E.4.3 Existing Risks Evolution . . . 160
E.4.4 New Risk Identification . . . 160
E.4.5 Risk Prioritization . . . 162
E.4.6 Risk Mitigation Plan . . . 163
E.5 Iteration 4 . . . 163
E.5.1 Risk Evolution . . . 163
E.5.2 Threshold of Success . . . 164
E.5.3 Existing Risks Identification . . . 164
E.5.4 Risk Prioritization . . . 165
E.5.5 Risk Mitigation Plan . . . 165

F Iteration Four Architecture 167
F.1 Iteration 4 Architecture Layer View . . . 167
F.2 Tasks . . . 168
F.2.1 Distributed Tasks Queue . . . 168
F.2.2 Sensor Endpoint Task . . . 169
F.2.3 Treat Paths Task . . . 170
F.2.4 Get Reading Task . . . 171

F.2.5 Match Way Subprocess 172
F.2.6 Process Reply Task 173
F.2.7 Compose Cache Objects Task 174
F.3 Routing Engine 175

Tables List

4.1 User Stories Prioritization - End User 41
4.2 User Stories Prioritization - Unauthenticated System Operator and Super User GUI 41
4.3 User Stories Prioritization - Super User 42
4.4 User Stories Prioritization - System Operator 43

5.1 Experiment 1 Average and Standard Deviation 66
5.2 Experiment 1 Results Distribution 67
5.3 Experiment 1 Results Distribution -2 67
5.4 Experiment 2 Average and Standard Deviation 68
5.5 Prototype Vs GraphHopper Results 76
5.6 Task Results 85

C.1 Quality Requirements Scenarios - Utility Tree 131

E.1 Risk Probability 152
E.2 Risk Impact 152
E.3 Risk Time Frame 152
E.4 Risk 1 153
E.5 Risk 2 154
E.6 Risk 3 154
E.7 Risk 4 155
E.8 Risk 5 155
E.9 Risk Exposure Matrix Iteration 1 156
E.10 Exposition to Risk Iteration 1 156
E.11 Risk Prioritization Iteration 1 156
E.12 Risk 1 Evolution Iteration 2 158
E.13 Risk 6 158
E.14 Risk Exposure Matrix Iteration 2 159

E.15 Exposition to Risk Iteration 2 159
E.16 Risk Prioritization Iteration 2 159
E.17 Risk 7 161
E.18 Risk 8 161
E.19 Risk Exposure Matrix Iteration 3 162
E.20 Exposition to Risk Iteration 3 162
E.21 Risk Prioritization Iteration 3 162
E.22 Risk 2 Evolution Iteration 4 164
E.23 Risk Exposure Matrix Iteration 4 165
E.24 Exposition to Risk Iteration 4 165
E.25 Risk Prioritization Iteration 4 165

Figures List

2.1 The OpenStreetMap Architecture . . . 12
2.2 Traditional Web Map Routing Implementation . . . 17

3.1 Gantt Chart 1 . . . 28
3.2 Gantt Chart 2 . . . 29

4.1 Traditional Web Map Routing Implementation . . . 33
4.2 Crossroads Initial Top View Architecture . . . 48
4.3 Crossroads Initial Component Architecture . . . 49
4.4 Docker - Linux kernel interface architecture . . . 55

5.1 Iteration one System Architecture . . . 61
5.2 Cologne sensor data source layered on the map . . . 62
5.3 Database Relational Model . . . 64
5.4 Web Routing Experiments System Architecture . . . 66
5.5 Incomplete map layer generated by the Geojson tool . . . 70
5.6 Map generated using OSRM match service . . . 72
5.7 Iteration Three System Architecture . . . 73
5.8 Routing results on a very congested route . . . 74
5.9 Prototype Vs GraphHopper Traffic Data Integration Demo Experience . . . 75
5.10 Iteration 4 Architecture Top View . . . 78
5.11 Iteration 4 Architecture Component View . . . 79
5.12 Unit Testing Results . . . 83
5.13 Integration Testing Results . . . 84

6.1 Browser-based client in map browsing mode . . . 91
6.2 Browser-based client in directions mode . . . 92

D.1 System decomposition . . . 134

D.2 Data Pipe Flow . . . 135
D.3 Database Updater System . . . 136
D.4 Compile and Update Routing Engine System . . . 137
D.5 Sensor Endpoint . . . 138
D.6 Validate . . . 139
D.7 Validate Message . . . 140
D.8 Analyse Attribute . . . 141
D.9 Compile OSRM file . . . 142
D.10 Prepare OSRM Map File . . . 143
D.11 Make Files Available . . . 144
D.12 Load New Map File to Router . . . 145
D.13 System Architecture - Layer View . . . 146
D.14 System Architecture - Component Diagram . . . 147
D.15 System Architecture - Components Architecture Diagram 1 . . . 148
D.16 System Architecture - Components Architecture Diagram 2 . . . 149

F.1 Iteration 4 Architecture Layer View . . . 167
F.2 Celery distributed tasks queue business process . . . 168
F.3 Sensor Endpoint Task business process . . . 169
F.4 Treat Paths Task business process . . . 170
F.5 Get Reading Task business process . . . 171
F.6 Match way business sub process . . . 172
F.7 Process Reply task business process . . . 173
F.8 Compose Cache Objects task business process . . . 174
F.9 Routing Engine business process . . . 175

Acronyms

AJAX Asynchronous JavaScript And XML. 61

AMPQ Advanced Message Queueing Protocol. 53

API Application Programming Interface. 11, 14, 15, 20, 22, 32, 57, 63, 71

BPMN Business Process Management Notation. 47, 80

BSD Berkeley Software Distribution. 57

CSS Cascade Style Sheet. 56

CSV Comma-Separated Values. 63, 64, 65, 69, 73, 86, 93

DNS Domain Name System. 21

EC European Commission. 4, 25

EIP-SCC European Innovation Partnership on Smart Cities and Communities. 4

EU European Union. 2, 3, 5, 9

FP7 Seventh Framework Programme for Research and Technological Development. 9, 16

GB Gygabyte. 18, 34, 65

GDP Gross Domestic Product. 3

GeoJSON Geographic JavaScript Object Notation. 61, 69, 71, 86

GIF Graphics Interchange Format. 15

GIS Geographic Information System. 32

GNU "GNU's Not Unix!". 24

GOS Goal Oriented Search. 19, 31, 32, 35

GPL General Public License. 24

GPS Global Positioning System. 11, 71

GSM Global System for Mobile Communications. 7

GUI Graphic User Interface. 16, 37, 39

HM Hierarchical Methods. 19, 31, 35, 36

HTML HyperText Markup Language. 47

HTTP HyperText Markup Transfer Protocol. 35, 46, 47, 50, 56, 60, 61, 71, 73, 76, 80, 84, 87

HTTPS HyperText Markup Transfer Protocol Secure. 51, 56, 82, 86, 87

ICT Information Communication Technology. 4, 5

IoT Internet of Things. 1

JOSM Java OpenStreetMap editor. 11

JPEG Joint Photographic Experts Group. 15, 18

JS JavaScript. 47, 56, 57

JVM Java Virtual Machine. 52

KB Kilobyte. 74

LRU Least Recently Used. 53

MB Megabyte. 74

MOM Message Oriented Middleware. 52

MVP Minimum Viable Product. 24, 74

MVT Model View Template Architectural Pattern. 53

NP Nondeterministic polynomial. 10, 20

OGC Open Geospatial Consortium. 15

OSM OpenStreetMap. 11, 12, 13, 14, 15, 17, 18, 19, 21, 22, 32, 33, 34, 35, 36, 54, 63, 64, 69, 74, 81, 86, 93

OSM2PGSQL OpenStreetMap to PostgreSQL. 12

OSRM Open Source Routing Machine. 19, 20, 31, 35, 37, 47, 50, 57, 59, 63, 64, 65, 68, 69, 71, 73, 74, 76, 77, 80, 81, 86, 87, 90, 91, 93

PDF Portable Document Format. 18

PNG Portable Network Graphics. 15, 18

PT Portugal Telecom SGPS, S.A.. 2

R&I Research & Innovation. 2

RAM Random Access Memory. 19, 34, 65

SME Small and Medium-sized Enterprises. 5

SONAE Sociedade Nacional de Estratificados SGPS, S.A.. 2

SQL Structured Query Language. 18, 54

SSL Secure Socket Layer. 51, 56

UNPF United Nations Populations Fund. 3

URL Universal Resource Locator. 16

UTC Coordinated Universal Time. 14

VCS Version Control System. 25

VGI Volunteered Geographic Information. 11

Wi-Fi Wireless Fidelity. 7, 8, 14

WMS Web Map Service. 15, 16, 34, 36, 37, 50, 64, 71, 86

WMTS Web Map Tile Service. 16

WSGI Web Server Gateway Interface. 51, 56, 81

XML Extensible Markup Language. 12, 18

YADPT Yet Another Django Project Template. 51, 53, 54, 55, 56, 77, 78, 81, 87

YAML Ain't Markup Language. 51, 55, 78, 81

Chapter 1

Introduction

1.1 Motivation

The growth of the Internet of Things (IoT) market, particularly when applied to Smart Cities, has allowed great improvement of its technological infrastructure which offered larger quantity and diversity of data, from parking occupation to traffic monitoring to measuring of pollution and air quality.

Ubiwhere was developing and installing its platform for smart cities named Citibrain - `http://www.citibrain.com`. This platform would collect, process, store and provide sensory information and web services related to mobility and environmental conditions. This information should be dynamically associated with the city's streets and roads in order to optimize city planning, event organizing, efficient routing and proactive incident response by city officials.

1.2 Objectives

Our internship, Crossroads: Real-time classification of roads based on the city data, was hosted by Ubiwhere. It aimed at studying how traffic data, received from sensors deployed over a metropolitan area, could be integrated into a Web Map Routing System. This integration would allow the system to calculate the shortest path between a source and a target location taking into account recent traffic information.

Project objectives:

- Study how traditional Web Map Routing Systems are implemented;
- Assess the feasibility of using real-time sensor data to update regularly traffic speed on a map;
- Identify and study tools and services that could integrate a possible system;
- Elicit Crossroad's functional and operational requirements;
- Reach a possible architecture;

- Develop and test a functional prototype;
- Have all produced artifacts accepted by the stakeholders by the end of the internship.

To be able to assess if the project had reached its goals and better evaluate the development process, we proposed several metrics and criteria that constituted the threshold of success for our project. Failure to achieve any of these goals immediately lead to the project to be deemed unsuccessful.

Threshold of Success:

- The Must Have User Stories as defined in section 4.3.1 of this document are developed and delivered by June 2017;
- The system respects all Constraints as defined in section 4.3.2 of this document;
- The system respects all Non Functional Requirements as defined in section 4.3.3 of this document;
- The workload is well distributed and tasks completed within the 1176 hours (42 ECTS) allocated to the internship;
- The process respects the proposed High-level Plan and Milestones with a less than 2 weeks discrepancy.

1.3 Ubiwhere

Based in Aveiro, Ubiwhere is a software company launched in 2007 currently developing projects in telecommunications, transport, tourism and smart cities.

> “ With a specific customers portfolio, from National Government to the majors telecom operators (such as Sociedade Nacional de Estratificados SGPS, S.A. (SONAE) and Portugal Telecom SGPS, S.A. (PT) Inovação), Ubiwhere's work has been noticed among these last years specially on Research & Innovation (R&I) European Projects, experience of the three founding partners, Since the beginning, we counted, with specialized know-how on high-tech products and services in areas such as telecommunications and next generation networks.
>
> R&I and user-centered solutions have been the hallmark of our growth, reflecting our culture of technology and shared ideas.We research and develop bleeding edge technologies, design state-of-the-art solutions and create valuable intellectual property to be an international reference in Smart Cities, Telecom & Future Internet” [Ubi16b].

1.4 Political and Social Context

In the past few decades, with the growth of urban areas, mobility and environment issues had become central to governments and citizens alike. The development of legislation and policies that could contribute to the sustainable development of our cities deserved a lot of attention by local, national and European Union (EU) officials.

1.4.1 Smart Cities

According to the United Nations Populations Fund (UNPF) [UNF16], by 2016, humanity was experiencing the largest rate of urban growth in history. More than half the world population lived in urban areas, by 2030 there would be about 5000 million people dwelling in towns and cities.

This kind of demographic pressure was bringing great social, economic and environmental strains to urban areas. Although this fast population growth opened opportunities for a whole new era of well being, efficiency and economic growth, it also represented a threat with high concentrations of poverty, the rise of economic inequality, criminality, environmental problems like traffic, air and water pollution, health problems related to the high population density, lack of access to clean water, sanitation, healthcare and education and inadequate, deteriorating and aging infrastructure.

With such high stakes, a great effort had to be made to guaranty welfare and sustainability of such rapid growing urban communities. Authorities should not only understand the issues at hand but should also be fed accurate information in order to legislate, manage, enforce and react accordingly. In order to solve problems and deal with threats, cities should become smarter [Cho+12].

Nevertheless obtaining a working definition of a Smart City was still a work in progress. Since it could incorporate governance, economic, political, demographic, sociological, infrastructural, technological and ecological and organizational factors there was not a single definition that could accommodate all these aspects. With such difficulty in defining what a Smart City Initiative should be it was also hard to define what factors were more important for its success.

1.4.2 Mobility and Transportation European Commission Policies

Urban Mobility was an important facilitator for growth, employment and the sustainable development of urban areas. In 2016, Traffic congestion in and around urban areas cost nearly EUR 100 billion or 1% of EU's Gross Domestic Product (GDP). Reducing traffic congestion, accidents and pollution had become a priority for all major European cities [Com17c].

To promote mobility that was efficient, safe, secure and environmentally friendly, the EU had created a land transport policy. This policy aimed at promoting efficient road freight and passenger transportation, fair conditions for market competition, promoting safer and more ecologically friendly technical standards and creating a harmonization

in fiscal and social policies between countries in order to guaranty the application of transport rules without discrimination [Com17a].

At this time, with the increase in freight and passenger road transportation, the risk of traffic pollution and road congestion was increasing. To prevent these risks, the European Commission (EC) was working towards a form of mobility that was sustainable, energy-efficient and respectful to the environment [Com17b]. To reduce the adverse effects of mobility, the Commission promoted co-modality by optimally combining various modes of transport within the same transport chain, technical innovations and a shift towards less polluting and more energy efficient modes of transport especially in the cases of long-distance and urban travel.

The use of technology to support road mobility could greatly enhance the sector by supporting a more efficient use of existing infrastructures as well as better managing transportation to guaranty a smaller ecological footprint [Com12].

1.4.3 European Innovation Partnership on Smart Cities and Communities

> “ European Innovation Partnership on Smart Cities and Communities (EIP-SCC) brings together cities, industry and citizens to improve urban life through more sustainable integrated solutions. This includes applied innovation, better planning, a more participatory approach, higher energy efficiency, better transport solutions, intelligent use of Information Communication Technology (ICT), etc.” [Com16].

This partnership had the objective of creating scalable and transferable technological solutions that could contribute to European Union’s 20/20/20 climate action goals to reduce high energy consumption, greenhouse gas emissions, poor air quality and congestion of roads.

EIP-SCC aimed to overcome hurdles impeding smarter cities development, to co-fund demonstration projects and to help coordinate existing projects and initiatives, by pooling resources together [SC16]. “It ultimately looks to establish strategic partnerships between industry and European cities to develop the urban systems and infrastructures of tomorrow. The Partnership follows the Smart Cities and Communities Initiative which was launched in 2011. This initiative initially only covered energy and had a budget of €81 Million, which grew to €365 Million and extended to include the transport and ICT sector with the launch of the Partnership in July 2012.” [SC16].

On its first Operational Implementation Plan draft [Eur14], the EIP-SCC had put great emphasis in Sustainable Urban Mobility, to promote change in Europe’s transport systems and the mobility habits of people and businesses, in urban areas, by creating solutions that concerned the creation of “an efficient and integrated mobility system that allowed for organizing and monitoring seamless transport across different modes”, “increasing the use of environmentally-friendly, alternative fuels” and “creating new opportunities for collective mobility” in order to create more eco-friendly mobility solutions and decrease the environmental impact of mobility.

To better achieve its objectives, this draft enumerated several potential actions to better address supply and demand, like the enabling of tools for seamless door to door

multimodality that would allow for the development of tools for ticketing and personalized transport planning, enabling faster, smoother travel, using different modes, optimizing traffic streams and minimizing energy consumption and traffic congestion [Eur14]. This document also addressed priority areas like integrated planning and management, knowledge sharing, open data, and standards.

By putting so much emphasis on mobility, sustainability, and ICT, this partnership aimed at " a significant improvement of citizens' quality of life, an increased competitiveness of Europe's industry and innovative Small and Medium-sized Enterprises (SME)'s together with a strong contribution to sustainability and theEU's 20/20/20 energy and climate targets". In order to achieve such goals, integrated, scalable Smart City ICT solutions especially in areas like energy, mobility and transport would be needed [Eur13].

1.5 Document Structure

The rest of this document is divided into the following chapters:

2. **Planning and Development Methodologies:** covers all planning activities taken, methodologies and tools used during this internship;
3. **Background Knowledge:** describes the necessary infrastructure to provide sensor data, explores potential map data sources and analyzes traditional web map routing implementations and the services that compose it;
4. **Preliminary Activities:** describes the initial exploratory work that was conducted in order to reach requirements and a possible architecture for the system;
5. **Development:** contains a description of the four iterations taken during the development phase, as well as an evaluation of resulting artifacts;
6. **Future Work and Conclusions:** addresses how the final prototype could be further developed as well as the conclusions that were taken from development.

1.6 Conclusions

The amount of attention and political interest towards smart cities and mobility issues made the Crossroads internship a valid project at this time. EU Funding available towards these initiatives constituted an additional motivation for the project.

We proceeded to gather background knowledge regarding our project. We started by studying what kind of infrastructure would be necessary to gather sensor data and transform it into usable information. Then, we took a look at possible map sources and tools that should allow us to manipulate map data. Finally, we studied how browser-based, web routing systems were traditionally implemented and what type of services they integrated.

Information gathered through this project phase allowed us to better understand whether or not the system we set ourselves to build was feasible and, if it was the case, what kind of components should be necessary to develop such system.

Chapter 2

Background Knowledge

2.1 Introduction

In the last chapter, we established the motivation behind our internship, the project objectives and a series of metrics and criteria that allowed us to determine its success.

This chapter will present the research made to identify potential data and map sources, possible tools and services that may be used by an eventual system.

We began by studying the necessary infrastructure to provide a potential system with the necessary sensor data. Since we wanted to use sensor data to regularly update traffic speed on a map, having a sensor network working properly was crucial to us. Then we assessed potential map data sources. Finally, we studied traditional web map routing implementations, tools, and services that could support our system.

2.2 The Infrastructure

The appearance of inexpensive and energy efficient sensors allowed the deployment of large urban sensor networks. Network Owners were able to monitor aspects of urban living like traffic, lighting or garden irrigation. The possibility of having small sensors with connectivity, embedded in everyday items further widened possibilities.

Developing infrastructures that could collect data transmitted by these sensors and transform it into valuable information was a huge challenge that could bring rewards to those who ventured into it.

2.2.1 Sensors

The integration of Wireless technologies like Global System for Mobile Communications (GSM), Bluetooth and Wireless Fidelity (Wi-Fi) into mass production communication devices like smartphones, personal computers and other appliances brought down the prices of these technologies due to mass production.

A whole new generation of processors and other electronic components with very low energy consumption allowed devices to run on batteries or remote power supply like solar

panels for a very long time without the necessity of maintenance.

These two technologies combined created a whole new generation of cheap sensors with a great autonomy that required no cable infrastructure or human intervention to operate.

These sensors could be distributed throughout cities or fitted into public transportation, police cars or other vehicles. These sensors collected large quantities of data and transmitted it to a central infrastructure that processed it into useful information. They could also operate autonomously controlling city lights, garden irrigation, and even traffic.

This system could help city officials to better manage, adapt and react. Better information could produce better use of resources and more informed decisions, thus improving the lives of city dwellers.

2.2.2 Real-Time vs Non- Real- Time Sensing System

A system with a lot of sensors, distributed over a large metropolitan area, represents a great challenge in terms of information processing. Accommodating a large sensor network and producing information in real time, would strain the system's scalability.

A real-time system should satisfy explicit bounded real time constraints to avoid failure. It should be consistent in terms of results and the time needed to produce them [KS15]. A very robust system would be necessary to respect these constraints.

Non-real-time systems could constitute a good alternative to real-time systems. They differ in the necessary effort needed to predict the response time and reducing it in a real time system. With a non-real-time system, we could simply schedule operations and eventually get the results. If we scheduled these operations on frequent enough intervals, we should have a near-real-time system[KS15]. Nevertheless, this system would still be non-real-time.

A real- time system required new events arriving at the system to be processed within a time interval and produced changes to be reflected in the system's knowledge base all at once. These restrictions could only be achieved in systems with low rates of incoming data and a small computational overhead.

When considering a distributed sensing system the size of a metropolitan area, we could easily conclude that guarantying scalability while working in real time would be virtually impossible.

When building systems based on inexpensive autonomous sensors, deployed far way from the data processing center and communicating through a third party network, it would be impossible to guaranty a maximum response time since it would be very difficult know how long it would take for data to reach the processing center.

2.2.3 BlipTrack Aarhus Case Study

The BlipTrack™[Sys17] system was deployed in the city of Aarhus in Denmark [Sys15]. It detected Wi-Fi and Bluetooth devices present in mobile phones and in-car audio and communication system. By identifying these devices in multiple sensors it extrapolated parameters like travel times, wait times and movement patterns.

Data collected by these sensors was transmitted using Ethernet or Mobile technologies and stored in data warehouses waiting to be processed.

By analyzing this raw data it was possible to extract valuable information regarding:

- Traffic queues and delays;
- Problem areas identification;
- The overall traffic capacity of existing roads;
- Identification of traffic patterns;
- Changes in traffic patterns.

This information allowed metropolitan authorities to better plan infrastructure, adjust traffic signs, lights and act proactively to minimize potential threats to circulation.

Raw data collected from this system has been made available by the CityPulse EU Seventh Framework Programme for Research and Technological Development (FP7) project [Cit16b].

This dataset is available at:

`http://iot.ee.surrey.ac.uk:8080/datasets.html`

2.2.4 Citibrain Platform case study

Citibrain [Cit17] was a three-company consortium headquartered in Aveiro, Portugal. It developed technological integrated products for smart cities, with the objective of improving cities and the quality of life of its citizens.

Citibrain aimed at deploying a network of low energy sensors with great coverage of the metropolitan area. This system communicated through a pre- existing infrastructure to supply data to the central ”brain” that processed it into useful information.

This information could be used to better manage the cities services like waste and traffic management and to improve the life of its citizens.

The companies that integrated the consortium were:

- Micro I/O [Mic17];
- Ubiwhere [Ubi16a]:
- Wavecom [Wav17];

This platform provided solutions in three main areas:

- **Environment**
  - **Smart Waste Management:** using data from sensors contained in garbage bins to better program garbage collection;

  - **Smart Air Quality:** collecting pollutant concentration data using small sensing stations installed in the current urban infrastructure to improve urban planning and quality of life of citizens;

- **Mobility**

  - **Smart Parking:** amalgamated all aspects of traffic management technology: vehicle detection, communication, informative boards, kiosks for payments, mobile app for drivers and an information system, into one integrated solution;
  - **Smart Traffic Management:** managed traffic in urban areas by analyzing data from sensors throughout the city and adjusting vertical signs, informative panels in critical points;

- **Payments**

  - **Smart Vending:** integrated monitoring, Internet transmission, and delivery of data from vending machines allowing its remote management;
  - **Smart Card:** monitored physical access to buildings, controlled closed spaces and substituted money in all transactions at those locations.

Citibrain Smart Waste: [Cit16a], had motivated the creation of Crossroads and could directly from it. This product used a series of sensors inside waste disposal bins to collect information about the present location, capacity, temperature and whether or not the bin was standing.

With the collected information, the system was able to determine which bins needed attention from the garbage collecting crews and devise a circuit that took into account not only the garbage bins as waypoints but also factors like road inclination, the garbage collection truck turn radius, traffic and road obstructions. This system made garbage collection and bin maintenance as efficient has possible, saving time and fuel.

Even though this product shared some similarities with Crossroads, the garbage collection problem was an entirely different problem. This product tried to solve the traveling salesman Nondeterministic polynomial (NP)-hard problem with several constraints added.

These constraints allowed the elimination of several roads in which the garbage collection truck would not be able to pass, would take long to traverse or demand too much fuel. Using the reduced map produced by these constraints and the information from the sensors in garbage bins, the routing engine should be able to calculate the best circuit for garbage collection.

Citibrain Smart Traffic: managed traffic in an urban environment. Static and mobile sensors collected data. This solution operated changes in traffic by manipulating vertical signs and informative panels deployed at critical points and by sending alerts to a mobile application.

2.3 Map Sources

Creating and maintaining maps was a very expensive and time-consuming endeavor. Since this activity was completely out of the project's scope, we needed to study alternative map sources that allowed us to obtain precise geographic information without having to spend resources or money. With this intent, we studied several alternatives.

2.3.1 OpenStreetMap

The development of the Internet and Global Positioning System (GPS) and its widespread integration into small devices like smartphones and other mobile devices not only created high demand for freely available spatial data but also became an important source of geographical information. This advancement boosted the availability of Volunteered Geographic Information (VGI) over the Internet.

Open source projects like the OpenStreetMap (OSM) [Fou16d] created and made available highly detailed maps based on VGI that could be exported into Vector Data Formats. This information, voluntarily made available by participants, was organized into a central database and distributed it in multiple digital formats through the World Wide Web [ZH12].

OSM was created in 2004, when map data sources were controlled by private and governmental authorities. The monetary and legislative barriers to access map information made it only available to large companies [Buc15]. This information was only available to the general public through the acquisition of GPS devices. These devices were expensive and required further monetary investment to be updated.

The original idea behind OSM was to create an open source editable map of the world based on a Wikipedia-like model. Instead of depending on private corporations and government authorities, this map would be constructed through the contributions of volunteers.

To be able to properly manipulate, edit and use the information provided by OSM there were several tools and projects made available to the community under an open source license:

- **The OSM website Rails Port [Fou16e]:** written in Ruby, this project contained the user interface and the proper Application Programming Interface (API) necessary to deploy the API main site;
- **Search and geocoding Nominatim [Qui17]:** this service allowed any set of geographic coordinates, address, or OSM identifier to be translated into one of the other two formats;;
- **Desktop map editor Java OpenStreetMap editor (JOSM) [Fou17c]:** written in Java, it was the most popular and powerful OSM editor available;
- **Online data editor iD [Fou16b]:** was a simpler map editor written in Javascript and executed in a web browser;
- **Default style at OSM.org [All17]:** was the default map style for OSM;

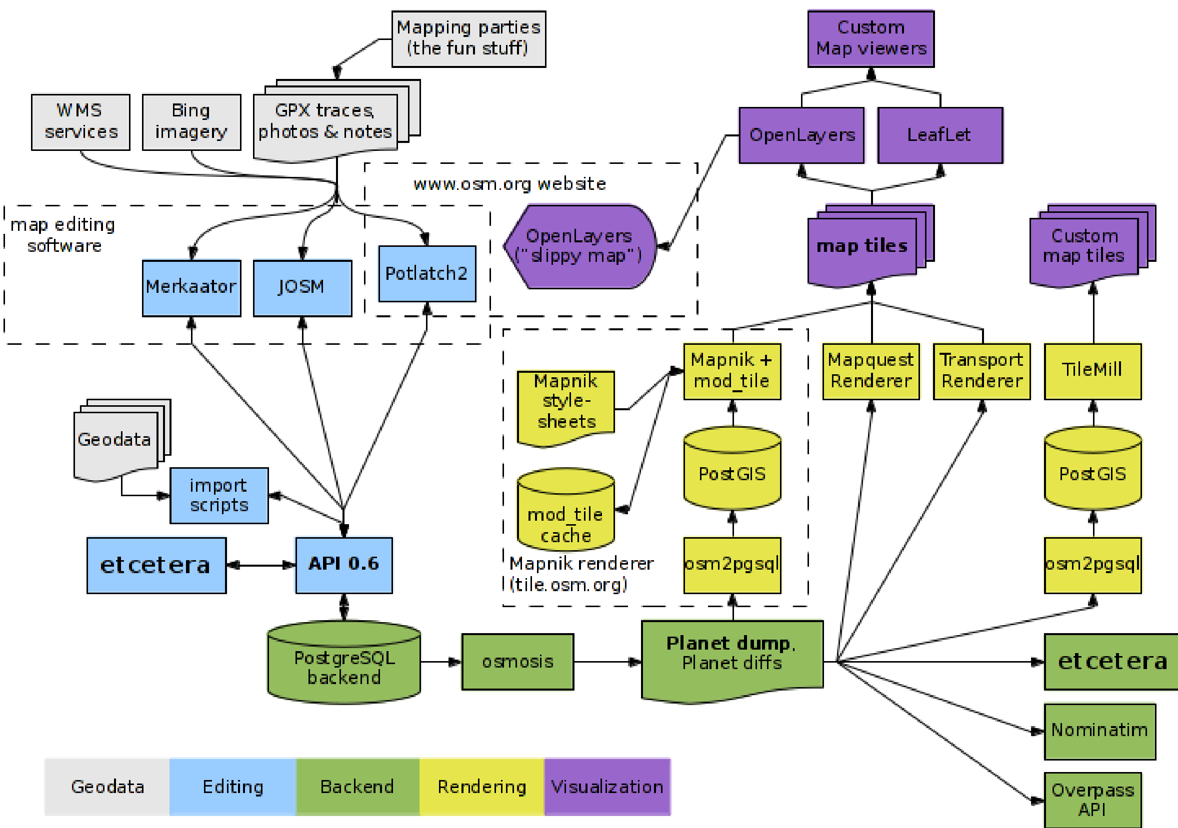


Figure 2.1: The OpenStreetMap Architecture
Source: `http://wiki.openstreetmap.org/wiki/Component_overview`

- **OSM data processing tool Osmosis: [Fou16f]:** allowed data, exported from the OSM website, to be imported into a local database and vice versa;

- **OSM Data Importer tool OpenStreetMap to PostgreSQL (OSM2PGSQL) [Fou17d]:** imported OSM Extensible Markup Language (XML) files into PostGIS[Fou16g] databases;

- **Slippy map library Leaflet [Aga15]:** "The leading open source JavaScript library for mobile-friendly interactive maps";

- **Mapnik [Pav16b]:** a backend tool that used the OSM map information to render maps.

Conceptual data model of the physical world: The OSM data model contained basic components called **Elements** [Fou16a]. There could be three types of Elements which had tags and attributes to give them meaning:

- **Node:** represented a point on the surface of the planet defined by at least an id number and a pair of geographic coordinates. It could be used to define standalone features or to shape ways;

- **Way:** an ordered list of between 2 and 2000 nodes that defined a polyline. It represented linear features such as roads, railways, and rivers or boundary areas (solid polygons) such as buildings or forests. In this second case, the first and last node was the same, they were called ”closed ways”;
- **Relation:** documented relationships between two or more elements (nodes, ways or relations) which typically had an assigned role. Relations could have different meanings defined by its tags. Each relation had at least a ”type tag”. These tags were interpreted together to make sense of which relation this element defined;
- **Tag:** described the meaning of the particular element it was attached to. They were organized into a ”key/ value” format. There was no fixed dictionary for tags although there were several documented conventions for its use;
- **Attribute:** represented a property from a given element which may or may not make it unique:
  - **id:** for identifying the element. There could be a Node, Way and Relation with the same id but never two elements of the same type;
  - **user:** displayed the name of the last user who modified this element;
  - **uid:** displayed the user id of the last user who modified this element;
  - **timestamp:** time and date of the last modification;
  - **visible:** a boolean attribute that defined the element visibility;
  - **version:** the edit version of the object. A new element had version number 1;
  - **changeset:** consisted of a group of changes made by a single user over a short period of time.

Semantic Objects were available to describe more complex objects. With them, it was possible to represent the geometry of the physical world. Semantic objects often used interchangeably with Elements which were a data primitive to represent Semantic Elements. This could cause some confusion.

There were four types of Semantic Objects:

- **Point**: defined a point in space;
- **Linear**: defined a linear feature;
- **Polygon**: defined simple or complex polygons and was usually used to express area boundaries;
- **Relational**: used to express how elements worked together.

This limited number of Elements and Semantic Elements along with an arbitrary number of Tags were used to describe any feature in a map along with its relations. They allowed for very complex maps to be rendered and used by other services like Geocoding or Routing servers to complete rather complex requests.

By crowd-sourcing geographic data, developing and maintaining the software tools necessary to build, use and maintain maps without any enterprise and governmental meddling, the OSM initiative encouraged the growth, development, and distribution of free geospatial data that anyone could use and share[Fou16d].

2.3.2 Google Maps

At the same time OSM was created, Google identified the necessity of a similar system and created Google Maps [Goo17a] which was launched in 2005. In 2008, Google recognized the importance of letting the community improve its maps and created Google Map Maker [Goo17i], nevertheless,at this time, all maps were still propriety of the company. Although it provided an API which should allow maps to be embedded on third-websites, the system was closed and regulated by Google Terms of Service [Goo16].

Even though the service was totally free (not the API though [Buc15]), it came at the cost of our privacy and the loss of control of what we could see on the map. The service chose what it considered relevant to us instead of displaying what was actually around. Google also used geographic search results and location information for marketing purposes.

Google Maps API Google Maps already possessed a Maps JavaScript API that implemented most of the services Crossroads would implement. It contained several HTTP web services as well as client libraries that allowed their use with several programming languages.

Web Services:

- **Geocoding API:** provided geocoding and reverse geocoding of map addresses and coordinates [Goo17c];
- **Google Places API:** featured many services like location awareness, search and retrieval of information about local businesses and points of interest, auto complete type ahead and location based predictions [Goo17f];
- **Elevation API:** provided elevation data for all locations on the surface of the earth [Goo17b];
- **Road API:** identified the roads a vehicle was traveling along and provided additional metadata about those roads, such as speed limits.[Goo17g];
- **Geolocation API:** Geolocation based on information given by cell towers and Wi-Fi nodes [Goo17h];
- **Directions API:** multi part directions for a series of waypoints. Directions for several modes of transportation were available as well as several traffic models that allowed us to estimate the predicted time in traffic based on historical averages. It was only available to clients with a Premium Plan client ID [Goo17e];

- **Timezone API:** received a set of coordinantes and a date and returned the name of the timezone, the time offset from Coordinated Universal Time (UTC), and the daylight savings offset [Goo17d];

Even though Google Maps API was very complete, its services were paid. We did not have access to certain features, like the traffic predictions, unless we subscribed a paid plan. This traffic prediction feature was based on historical averages rather than data recently collected from a sensor network.

2.3.3 Choosing a Map Source

Taking into account the alternatives, we concluded that there was no real substitute to OSM for our project. Since it was a crowd-sourced, free initiative, it made our system independent of any license agreement with a private corporation that could change and imply costs or make our system dependent from an external corporate policy.

OSM had the added advantage of including a large number of side projects that developed totally free open source tools. These tools should prove valuable if we needed to manipulate map information or deploy services during our service development.

2.4 Standards: The Open Geospatial Consortium

The Open Geospatial Consortium (OGC) [Con16c] was an international nonprofit organization dedicated to creating open, freely available standards to improve sharing of geospatial data. These standards were used in a wide variety of domains including Environment, Agriculture, Health, Meteorology, sustainable development, among many others.

The OGC implementation standards differed from abstract specifications. They targeted a more technical audience to detail the interface structure between software components.

”An interface specification is considered to be at implementation level of detail if, when implemented by two different software engineers in ignorance of each other, the resulting components plug and play with each other at that interface” [Con16b].

2.4.1 The Web Map Service

The Web Map Service (WMS) [Con17b] was a standard protocol that dynamically produced digital image files, suitable for display on a computer screen, of spatially referenced data from geographic information. These ”maps” were usually rendered in pictorial formats such as Portable Network Graphics (PNG), Graphics Interchange Format (GIF), Joint Photographic Experts Group (JPEG) or as vectoral-based graphical elements.

This protocol standard defined three types of operations that return different resources:

- Service level metadata;

- A map rendered into a digital image in which dimensional and geographical parameters were well defined;
- Information about particular features shown on the map.

These service operations were invoked by submitting requests using Universal Resource Locator (URL). If two or more images were produced with the same geographic parameters and output size in a format that supported transparent backgrounds, these maps should be correctly overlaid to produce a composite map. These characteristics allowed for the creation of a network of distributed map servers from which clients could build highly customized maps.

WMS applied to instances that published their ability to produce maps rather than store map tiles. A basic service classified its geographic information holdings into Layers and offered a number of predefined styles in which to display these layers.

This protocol standard was described in the OpenGIS® Web Map Server Implementation Specification [La 06].

2.4.2 The Web Map Tile Service

The Web Map Tile Service (WMTS) [Con17a] protocol standard was built on earlier efforts to develop web services for cartographic maps distribution. It provided a complementary approach to WMS for tiling maps. While WMS focused on rendering custom maps, FP7 traded the flexibility of custom map rendering for the scalability possible by serving static data where the bounding box and scales had been constrained into discrete tiles. The use of static data elements also allowed for the use of local caching that further enhanced scalability.

This protocol standard was described in the OpenGIS® Web Map Tile Service Implementation Standard [MPJ10].

2.5 The Traditional Web Map Routing Implementations

To implement a web map routing system, several components were required. The system was constituted by a client application or a client running on a web browser and three services that replied to the client's requests with the necessary information.

These services were:

- **Web Map or Tile Service:** Responsible for rendering or storing the necessary map tiles to answer the client's requests. These tiles allowed the client to present a map to the user in its Graphic User Interface (GUI);
- **Routing Service:** calculated a path from a source to a target location, according to the parameters present in the client's request. The path, returned to the client application, was a polyline that was layered on the map and presented to the user

in its GUI. This service also supplied directions in different languages and several routing options;

- **Geocoding Service:** Responsible for translating an address, geographic coordinates or an *OSM_id* into one of the other two types of data according to the request. This service was necessary for the client to be able to find points in the map, translate points in the map into a set of coordinates, or to translate the source and destination of a route into a set of coordinates or *osm_id* that the routing service could understand.

By layering maps, polylines, and other components, the client's GUI was to present the user a rather complex representation of the map and other features like points of interest or routes.

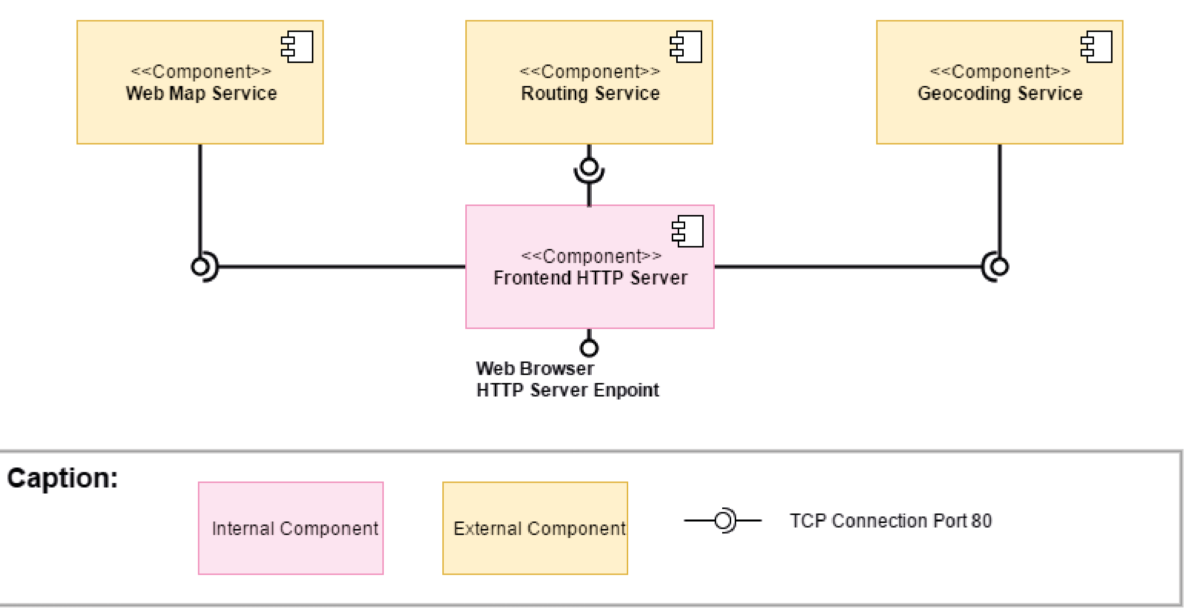


Figure 2.2: Traditional Web Map Routing Implementation

2.6 Web Map or Tile Services

To work properly, raw geospatial vector data was rastered into tiles that could be combined to produce a visual map. While web mapping services generated the necessary map tiles on the fly, tile mapping services generated and store them previously.

2.6.1 Mapnik Map Rendering Software tool

Mapnik [Pav16a], written in C++ with Python binding, was by far the most popular open source map rendering toolkit. It rendered layers that were used to construct the

web map on the OpenStreetMap Website, most of the other web map services, that used OSM maps, also used this tool to render their maps.

It supported a variety of data formats and styles. Although it was more common to load those data files into a PostGIS database and access them via Structured Query Language (SQL) queries, it could process OSM XML files directly. The Shapefile vectorial data format [Ope16] was also usually used to assist in map rendering, especially geographical features like the coastline.

Mapnik could output map imagery in a variety of file formats: PNG, Portable Document Format (PDF), JPEG , among others. The primary use of this rendering tool involved rendering thousands of 256x256 pixel tiles that could be combined and layered to display the desired map.

2.6.2 Deploying our own OpenStreetMap Server

The official OSM web map server operated in donated machines, it should be used in moderation [Fou16h]. For more intensive use or commercial purposes, we should simply deploy our own instance directly from packages [Fc13b].

By deploying our own instance, we could control the information was displayed in our maps and their aspect. Nevertheless, generating and serving our own tiles would require a robust infrastructure, especially if we would be covering a very large geographic area.

"In general, requirements will range from 10-20 Gygabyte (GB) of storage, 4GB of memory, and a modern dual-core processor for a city-sized region to 300GB+ of fast storage, 24GB of memory, and a quad-core processor for the entire planet"[Fc13a] .

2.6.3 Outsourcing the Web Map Service

A good alternative to deploying our own OSM server would be acquiring the service from a third party. Since most companies offered a Tier based pricing model, we could tailor the service to our needs and benefit from not having to maintain our own infrastructure, at the cost of less control and optimization of the information displayed and the aspect of our tiles.

A list of companies that offer consulting, tile-hosting or other services for OSM:

`https://switch2osm.org/providers/`

2.6.4 MapProxy

MapProxy [KG15] was an open source proxy server for geospatial images. This service allowed significantly increase performance against using the OSM services alone if the same general area was used intensively.

2.7 Routing Services

The key component to a Web Map Routing implementation was its Routing Service. This service received at least two sets of coordinates (source and target location) and an arbitrary number of parameters from the client and returned a data structure representing a polyline and optionally a set of directions that could help the client to navigate the area from the source to the target location.

Annex A presented a brief study of the most common algorithms used by routing engine. Computing the shortest/ fastest path between two locations was a very intensive operation. Therefore, numerous were put into place to make sure the number of visited nodes during the search was as low as possible. Nevertheless, the worst case scenario was maximized by an $O(n^2)$ function.

There were several open source implementations of routing services. Some used Goal Oriented Search (GOS) techniques, others Hierarchical Methods (HM) or both. Since a Routing Service had to reply to a great number of requests per second, if we did not need to make regular changes to the graph and edge weights, HM were preferable to GOS, since the search was conduct over a very condensed sub-graph at a higher hierarchy.

The two open source services described below use primarily Hierarchical Methods:

2.7.1 Open Source Routing Machine

Open Source Routing Machine (OSRM) [LV11] was a routing server designed to use OSM as its source of information. This service used HM instead of the more common A* algorithm to compute the fastest path between two given locations on a map. These alternative algorithms made this server considerably faster when calculating paths in a very large map.

To run the routing service, the contracted hierarchy map was pre-calculated from an OSM file export. The first compilation step extracted map information into a highly normalized format. During this step, a vehicle profile, that represented the typical behavior of a certain mode of transport, produced extra restrictions that allowed the map to be smaller. This resulted in a better performance at the cost of having to pre-calculate a map for each mode of transport available. This initial operation could be very time consuming (it could take hours for the entire planet).

After this initial step, data should be compressed using a hierarchical algorithm to create the necessary files for the server to be deployed. The resulting files were highly optimized and able to handle continental sized networks in a matter of milliseconds.

Although more efficient, this process brought several disadvantages:

- The necessity of recompiling the map file each time a new OSM file became available;
- The necessity of maintaining a different file and service for each given profile;
- The inability of editing the map as new traffic information became available.

Running OSRM with shared memory: Shared memory allowed us to share data among several running processes and make data stored in the shared block of memory made persistent. This feature brought great advantages in high availability environments where several instances of OSRM could share a compiled version of the OSM map loaded directly into the Random Access Memory (RAM). Furthermore, by using a process independent data management tool, it was possible to load and replace data sets without any downtime or noticeable delay. New data was loaded into a separate memory region, and processes were notified of the new data availability after completion [Lux13]. Finally, since data was loaded independently, any failing process could be substituted in significantly less time further contributing for the system availability [Lux13];

Traffic Updates: OSRM featured experimental support of traffic data since version 4.9.0. This support was achieved by providing an additional file featuring edges and its max speed update during map compilation [con17b];

Turn Penalty data: OSRM also supported penalties applied to turn maneuvers for more realistic modeling. This feature was also achieved by providing an additional file containing information regarding each turn [con17b].

The OSRM HTTP API [con17a] could provide several services:

- **Route:** returned the fastest path between the coordinates supplied by the request;
- **Nearest:** received a set of coordinates to the city network and a number parameter. Returned the nearest n nodes where n is the number parameter;
- **Table:** Returned the duration of the fastest route between all pairs of supplied coordinates;
- **Match:** tried to match a set of given coordinates to the existing road network in the most plausible way;
- **Trip:** Given a set of coordinates, it tried to solve the traveling salesman problem NP difficult problem;
- **Tile:** Generated vector tiles that could be viewed in a vector-tile capable slippy-map viewer.

2.7.2 GraphHopper Routing Engine

GraphHopper [Kar16a] was a routing library and server written in Java. It was designed to operate with desktop and mobile clients (Android and iOS). It was able to use several shortest path algorithms like Dijkstra's, Goal Oriented Search (flexibility mode) and Hierarchical Methods (speed mode).

The speed mode was much faster and less memory intensive at the cost of a large precalculation step before the service could be deployed to calculate the map graph. Since vehicle profiles were used to pre-calculate the graph, this mode was less flexible. Several profiles could be included at the same time.

Important features like real-time changes to edge weights, alternate routes and turn restrictions were only available in the flexibility mode. This project used the Maven build automation tool which made it easy to build and deploy it on a given machine. It was is supplied under an Apache Licence 2.0 [Lic04] and was capable of operating with Android clients. It constituted a very attractive alternative to existing Android navigation software.

GraphHopper Traffic Data integration [Kar16b]: This project integrated real-time traffic data into the GraphHopper routing engine. Since speed mode required map recompiling, it would not be possible to use traffic integration in real time for larger maps. It incorporated a web service that received requests to replace the max speed allowed in at least one edge of the map graph. By manipulating the maximum speed allowed it would be possible to simulate traffic effects in real time.

2.8 Geocoding Services

People and machines treated addresses quite differently. While people interacted with and memorized a street address easily, machines found it easier to use a set of geographic coordinates or an id to define a geographic location. In order for our system to work properly, we should translate street addresses to coordinates and vice versa. In a very similar fashion to the Internet and its Domain Name System (DNS) service, we would need our own geographic address translation service, the Geocoding Service.

This service performed two main operations:

- **Geocoding:** was the process of translating a street address into a set of geographic coordinates;
- **Reverse Geocoding:** was the process of converting geographic coordinates into a human-readable address.

2.8.1 Nominatim

The Nominatim [Fou16c] service was a free open source tool provided by the OSM foundation that was able to search OSM data by name or street address and translate it into synthetic addresses of *osm_id* and *place_id* points or geographic coordinates, and the other way around. This service offered three main operations:

- **Search:** Generated geographic coordinates, *osm_id* and *place_id* points from a street address or name;
- **Reverse Geocoding:** Generated an address from a set of geographic coordinates;
- **Address Lookup:** Looked up the addresses of up to 50 specific OSM node, way or relation and returned its street addresses.

Since this service was able to use and generate *osm_id* and *place_id* points, it was specially fitted to work with the OSM conceptual data model. It should be much more efficient to search for a specific element in a database for its *osm_id* and *place_id* than for a set of geographic coordinates.

This service used a PostGIS [Fou16g] database to store its information. After the service installation, an import and index of OSM data step was needed. This step was very time consuming and computationally intensive (it could take several days for the whole planet). The larger the area to be imported, the more computational resources would be needed for the service to run smoothly. This was taken into account when deploying the service.

2.8.2 Outsourcing the Geocoding Service

As a good alternative to deploying our own Nominatim Server, we could acquire the service from a third party. "Several companies provide hosted instances of Nominatim that you can query via an API, for example, see MapQuest Open Initiative, PickPoint, OpenCage Geocoder or LocationIQ" [Fou16c].

2.9 Conclusions

In this section, we initially studied how a network of cheap sensors deployed along an urban could be used to retrieve data that could be transformed into valuable information for city dwellers and local authorities.

Then we proceeded to assess possible map sources for our project, possible web services and software libraries that could help us in our endeavor. We quickly came to the conclusion that Google Maps was not adequate for the task at hand and there was no real alternative to OSM.

After establishing OSM as our map source, we proceeded to study how traditional Web Map Routing Systems were implemented. We studied Web services associated with them, the alternatives we had and whether or not we could simply outsource the service.

At this point, It was our conviction that the choice of the Routing Service would be central to our project and that we would need to figure out how to feed traffic information to this service in order for our project to succeed.

Reaching our objectives in 10 months without a devising a high-level plan or development methodologies would be impossible. To better guide us through the internship, we would schedule activities. We would also have a better assessment of work progress at all times. The next chapter describes all methodologies and activities planned.

Chapter 3

Planning and Development Methodologies

3.1 Introduction

In the last chapter, we described the sensor infrastructure and map sources necessary to support an eventual system. Then we assessed services and tools that could help us devise this system.

In this chapter we will describe methodologies and tools used to better plan and assess the progress of our project. We will also describe the project high-level plan.

We started by studying methodologies and tools that would allow us to better plan, manage and account for our activities. Then we proceeded by developing and high level plan and schedule for the whole project. Even though there was great uncertainty regarding how our project objectives would be met, this high-level plan would guide us through the whole process.

3.2 Methodology

During the first semester, activities followed a rigid sequence. Therefore a Waterfall life cycle [Roy70] was used for this part of the project. Each phase of this early part of the project was self-contained and followed by the next activity in an orderly manner.

For the second semester, since the objective of the project was to study how to integrate the information provided by sensors into a web map routing system, there was great uncertainty.

Nevertheless, eliciting initial high-level requirements and reaching an acceptable architecture specification for our system was relatively easy. Therefore, in order to quickly evaluate if the architecture specification met the elicited requirements and quickly address any potential risks to the project, we adopted a risk driven Spiral Life Cycle using an Iterative Design Model [Boe88].

By putting great emphasis in setting objectives, identifying and resolving potential risks early on, we were able to quickly prototype, test, analyze and refine our system

during each iteration.

With every iteration, we developed the Minimum Viable Product (MVP) [Rie11] with just enough features to allow us to begin learning as quickly as possible.

We had a good idea about requirement satisfaction and risk mitigation after the MVP's evaluation. If the prototype did not satisfy all requirements or new risks had arisen, we could prepare a new iteration and study alternatives. This process repeated itself until achieving stakeholders satisfaction or time ran out.

By having the MVP so early, we were able to evaluate it, focus on mitigating risks, study possible alternative and were able to accommodate change into the process and adapt right from the start.

At the end of the process, we obtained a prototype that met all operational requirements, complied with all must have User Stories and could be further developed into an actual product. Lessons learned from this process could be applied to other projects like Citybrain's Smart Traffic and Smart Waste that had served as an inspiration for this project.

Finally, in order to facilitate continuous integration and delivery, several techniques were be adopted:

- A single source GitLab repository was maintained;
- Progress was merged with the master regularly;
- All testing was be conducted in a sandbox that simulated the production environment;
- Every stakeholder was able to observe what was happening and get the latest scripts using Gitlab and Redmine.

3.3 Tools

During development, several tools were used to manage and control the project, the repository and develop the necessary code.

3.3.1 Redmine

"Redmine was a flexible project management web application. Written using the Ruby on Rails framework, it wass cross-platform and cross-database.

Redmine was open source and released under the terms of the "GNU's Not Unix!" (GNU) General Public License (GPL) v2" [Lan14].

Redmine featured:

- Issue tracking system;
- Gantt chart and calendar;

- News, document and file management;
- Feeds and email notifications;
- Time tracking;
- Custom fields for issues, time-entries, projects and users;
- Git integration.

Using Redmine, we were able to manage issues and high level tasks as well as the amount of time spent on them.

3.3.2 GitLab

Git [Git16a] was an open source Version Control System (VCS) that allowed multiple users to work, track issues and manage changes in computer files. One of the main features of Git was the fact that it was distributed allowing users to clone entire repositories and have a personal backup of the repository. This backup could be further branched allowing an unlimited number of workflows that could be merged back again if necessary, allowing user to work through discrepancies and other issues that could arise.

GitLab [Git16b] was a web based Git repository manager that offers all its available features from a graphic user interface .

3.4 First Semester Planning

With a project this large lasting for 10 months, it was of the utmost importance to try to plan activities and estimate how long each activity was going to take, in order to bring some order into such an exploratory process.

The Gantt Chart in section 3.6 presents how time was spent in the various activities.

During the first semester, activities were planned sequentially in a very rigid model. Since we did not have much information about what kind of system we were going to develop, we had to understand the context behind such project and study existing solutions and tools. With that knowledge, we were able to enter a preliminary work phase that allowed us to get a better idea of what our system was supposed to do and what requirements and constraints should guide its operation. Finally, we designed a possible architecture.

Activities Description:

1. **Get acquainted with Ubiwhere:** during the first week, we met the rest of the team and learned about the company and the tools used to manage its projects and repositories;
2. **Validate the idea:** understanding the reasons to undertake such a project, the EC's policies that drove mobility and the overall scope of the project took the initial

weeks. After assessing why Crossroads was a valid idea, we tried to determine how we could build it by studying routing algorithms, open source tools that could help achieve our goals and existing routing systems from which to draw inspiration;

3. **Background Knowledge:** the idea validation activity left us with the information necessary for us to write this part of the report. This activity continued throughout the semester as new information became available and the project evolved;

4. **Preliminary Work:** using the tools and technologies discovered during the idea validation activity, we put great emphasis into deploying and testing these solutions by ourselves. This activity allowed us to understand how we could build a potential prototype;;

5. **System Requirements:** first, we wrote User Stories that represented high-level representations of functional requirements. Then, we elicited constraints and non-functional requirements that the system should follow in order to operate properly;

6. **System Architecture:** we wrote scenarios to refine quality requirements. Then we looked at architectural patterns. Finally, we developed a high-level architecture of our system;

7. **Write the Intermediary Report:** documentation was the most important part of the process. It began during the idea validation phase and was extended up until the delivery deadline.

3.5 Second Semester Planning

After eliciting initial requirements and defining a possible system architecture, we had a clear picture of what we needed to accomplish. Nevertheless, there was still great uncertainty and further testing was needed.

To obtain more information and test our initial system specification, we identified and dealt with the most urgent risks. Taking into account the risk mitigation plan, we built a small prototype that allowed us to evaluate our specification and make the necessary adjustments. We iterated this process until we got a system that satisfied the stakeholders.

We planned four monthly iterations to refine our system specifications and prototype.

The Gantt chart in section 3.6 presents the estimation of time that was spent on each of the planned activities.

Activities Description:

Even though we had a clear image of what a full system should look like, we had to prove whether or not many assumptions made during the preliminary work phase were correct. To achieve these goals, we entered a completely different development phase.

From this point on we focused on mitigating risks by quickly devising small experiments and developing a series of small prototypes to prove the feasibility of our initial architecture.

With this in mind, we planned four monthly iterations. Each iteration was composed of five phases:

1. **Planning:** devising a high-level plan of what we should do during the iteration;
2. **Risk Analysis:** risk identification, prioritization and the preparation of a mitigation plan;
3. **Engineering:** taking into consideration the high level and risk mitigation plan, we updated requirements and architecture, engineered components and planned experiences;
4. **Construction:** artifact development and experiments conduction;
5. **Evaluation:** artifact testing and experimental results analysis.

The activities developed during each iteration as well as its results are better described in chapter 5

3.6 Gantt Chart

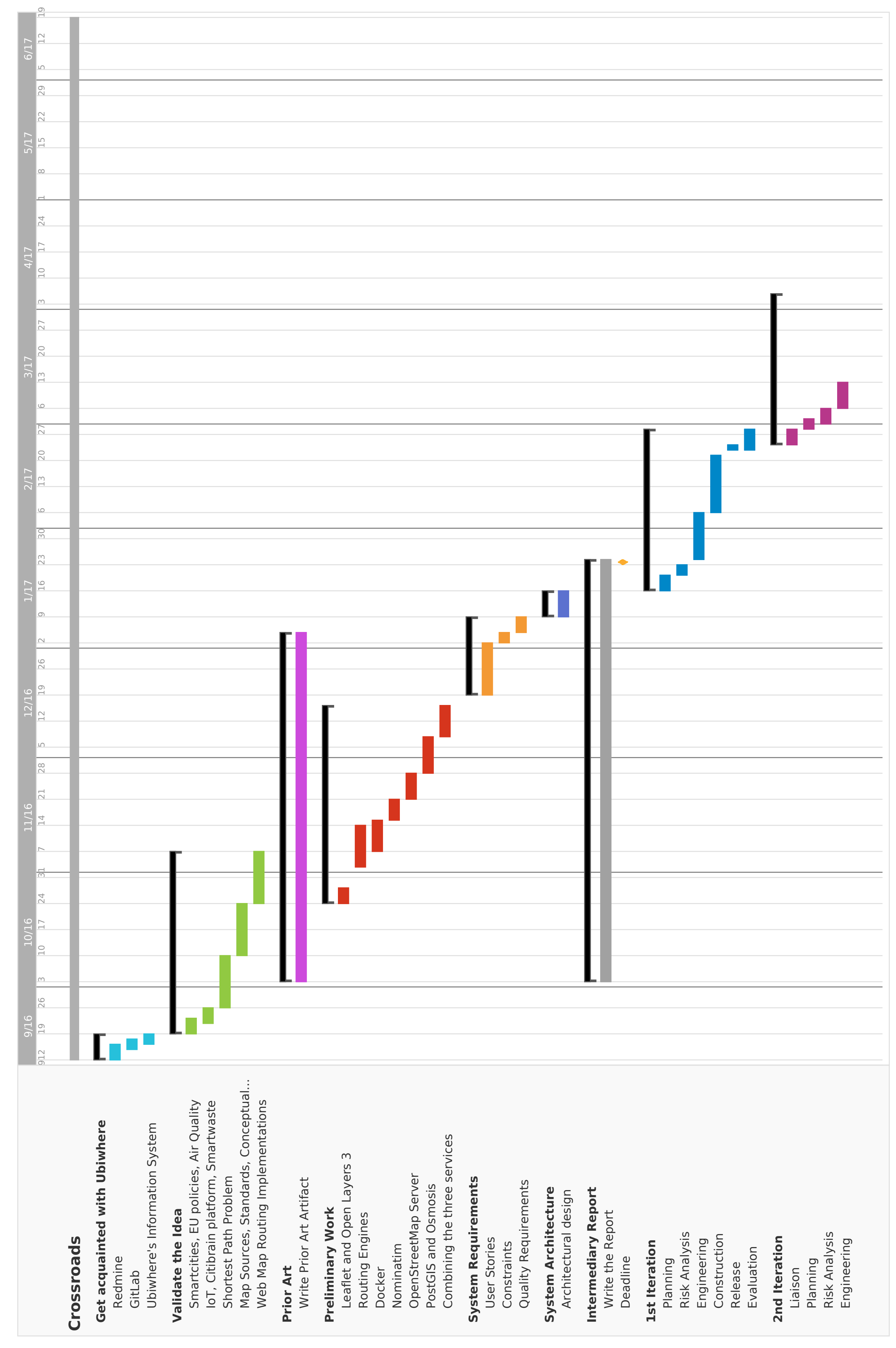


Figure 3.1: Gantt Chart 1

Figure 3.2: Gantt Chart 2

3.7 Conclusions

By developing a high-level plan and studying tools to manage it, we were able to manage the whole process and assess our progress.

To better understand what functional and operational requirements our system would have to observe, and what a possible architecture should consist of, it would be necessary to further analyze possible solutions in a preliminary work phase described in section 4.2.

Chapter 4

Preliminary Activities

4.1 Introduction

In the last chapter we devised a high-level plan and the studied tools to manage it.

This chapter describes the work that was conducted after the initial assessment of the possible infrastructure and technologies used on similar projects. With all the additional knowledge gathered during this initial phase, we were able to elicit functional and operational requirements for our project and reach an initial architecture.

Since this project was exploratory, this initial architecture was only meant to guide us through the initial phases of the development process and was expected to suffer changes as more information became available.

4.2 Preliminary Work

4.2.1 Introduction

After studying existing routing algorithms and solutions it was possible to enumerate a set of preconditions that should allow the proposed system to be built.

4.2.2 Choosing Routing Web Services

By analyzing the possible routing algorithms, we concluded that the solution's routing service would represent most of the necessary computational effort. Therefore, strategies had to be put into place, in order to make the routing service as economic as possible. There was no real alternative to Dijkstra's algorithm which was majored by an $O(n^2)$ complexity. The existing optimizations only produced faster results because they reduced the number of visited nodes, they did not improve the worst case scenario.

Using HM, although it added a computationally intensive precalculation step, produced much faster results and used much less computational resources when calculating the fastest path between two points as less nodes had to be visited, in order to find an answer to the problem.

Modern routing services that used HM, like GraphHopper and OSRM, could answer requests of transcontinental routes in a few milliseconds. GraphHopper offered the possibility of deploying the service using GOS (Flexibiliy mode) or HM(Speed mode) in order to compare how much faster the speed mode was. Therefore, any routing service deployed should use HM optimization.

The only real advantage of using GOS would be the fact that, as there was no pre-compilation step and data could be updated in real time using live sensor data. Nevertheless, since and eventual system would be based in inexpensive sensors deployed over a large metropolitan area and communicating through a vastly unknown network, we could not guaranty that the system would react deterministically within a given time interval. Therefore, there was no real advantage in using GOS.

4.2.3 Data Sources

One of the key elements of the proposed system were its data sources. The system depended on quality data to be able to operate properly.

Map data

After studying possible map data sources it was clear that there was no viable alternative to OSM.

Proprietary solutions, like Google Maps, did not constitute a viable data source, as the information given was conditioned by enterprise policy and most of them were paid or only available through a proprietary API. OSM constituted the only free Geographic Information System (GIS) data source available that processed detailed data necessary feed our system's necessities.

Nevertheless, fulfilling our GIS data necessities with a crowd-sourced solution posed several major issues. There was no real way of verifying data accuracy and, although all geographical data regarding certain area elements was present, the system that classifies those elements using tags was often incomplete, or there were keys that our system would not recognize because there was no tag dictionary and each contributor was free to classify the elements as he saw fit.

Since our system would only be as good as the data that fed it, considerable effort had to be made in order to complete missing data and try to standardize existing tags.

Sensor data

At this point, sensor networks that could support our system were still experimental and only covered a small part of the metropolitan area. Sensor coverage was also not total in analyzed data sets.

This created some issues when trying to calculate the fastest route. If a road that was covered by the sensor network was congested, the system would divert the route into nearby roads that might not be covered by sensors. Even though, with the available data and from an algorithmical standpoint this action would be correct, we would not know if this road was jammed because our sensor coverage was limited.

As with the map data, our system would only be as good as the sensor data that fed it. Without a good sensor data source the system would be little more than an overcomplicated routing application.

4.2.4 Web Map Routing Implementation

As it was described in chapter 2, this type of system would require three different services to be implemented. An architectural diagram of such a system can be seen in figure 4.1.

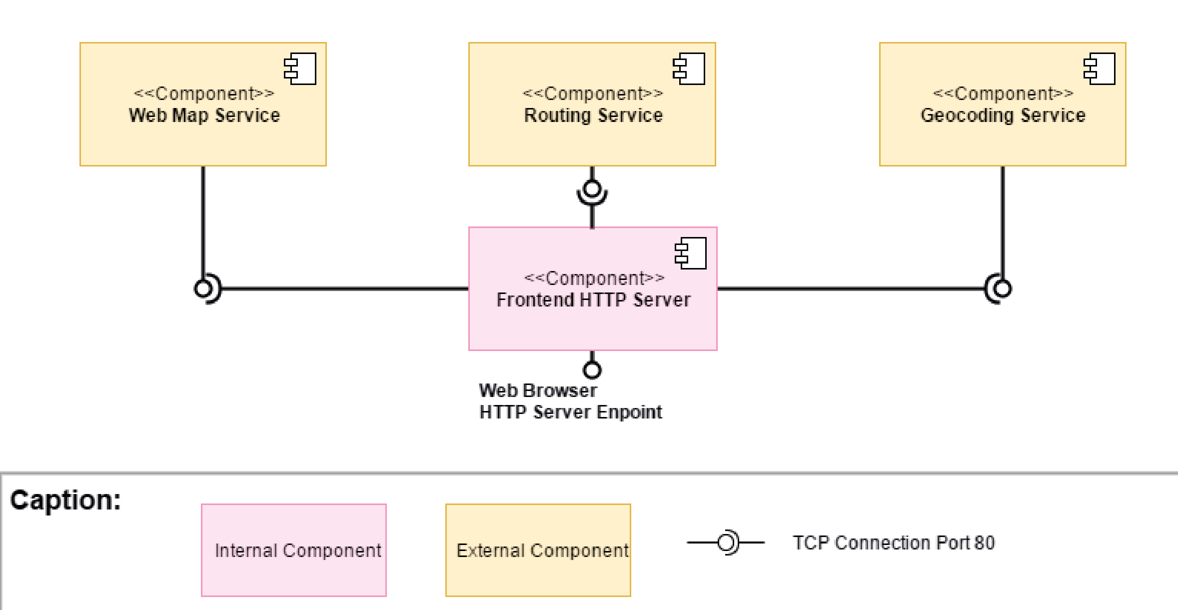


Figure 4.1: Traditional Web Map Routing Implementation

These services could be owned by the service operator or outsourced. There were several web services that provided the necessary services for free up to a certain number of daily requests. This fact allowed our prototype to operate without having to deploy these services ourselves. Nevertheless, outsourcing services posed some major disadvantages. We would not have control over the map data that was being used by those systems and could suffer from potential changes in the service licence agreement.

Since all of these services were highly complex there would not be enough time or the necessary skill set to develop them. There were several open source implementations of these services that could be modified, nevertheless their licence agreement obliged the company to make these modifications available to the community.

All these services used OSM maps as the source of information whether contained in a file or in a database. Therefore, a more attractive solution to make the service sensible to sensor data would be to simply modify the information they used to provide the service.

The traffic example could be used to illustrate this strategy: If the sensors deployed in a given street calculated the average speed of cars passing in it, we could use that

parameter to update the ”maxspeed” tag contained in the element that represented that street in the map.

Another example of this strategy could be achieved by manipulating the Restrictions tag group. An example of these tags would be the ”foot” tag that could have the value ”no” to prevent pedestrians from using the element. This group represented prohibitions of usage for certain vehicles and under certain conditions. Restrictions are described in the link below:

`http://wiki.openstreetmap.org/wiki/Restrictions`

If we manipulated these restrictions in the map, according to data received by sensors, we could modify the circulation conditions.

The downside of this strategy would be the necessity of owning the service, in order to be able to deploy these modified maps in it. Nevertheless, if this modifications did not produce alterations to the map elements, only the web routing service would have to accommodate this alterations and the other two system can be outsourced.

Web Map or Tile Service

The OSM WMS was already deployed in donated servers. This service proved adequate for the needs of our small prototype. Nevertheless, any further developments or production system would have to include our own WMS service or outsource the service, as the quantity of requests allowed by the donated server was very limited.

In order to test the feasibility of deploying our own WMS service, we were able to deploy the OSM Server into a Linux based virtual machine with 4GB of RAM. Although the service worked properly, it was a bit slow and sluggish to respond. Proxying the service made it considerably more responsive.

Since we imported an OSM export file containing the map of Portugal into the PostGIS, this might have had some influence into the response time. If we had limited ourselves to a city or metropolitan area map size map, we might have gotten better results using such a modest virtual machine.

Several WMS implementations were available at the Docker Hub but none was tested.

Geocoding Service

Like the WMS service, the Nominatim geocoding Server from the OSM Foundation was also deployed in donated servers. This posed severe limitations in terms of the number of requests that could be answered (1 request per second). Although this would be adequate for our initial prototype, we would need to deploy our own Nominatim service our ”buy” the service from a third party for a more complex system to run properly.

To properly test the service, the Nominatim server was deployed in a 4GB Linux Based Virtual Machine twice. One from the source code provided by the OSM Foundation and another from a Docker Container. As expected the container provided an automated deployment was easier to install.

In order to properly test the Server, an OSM export file containing the map of Portugal was imported and indexed into the PostGIS database. It took nearly 4 hours to

complete the job as expected on such a modest virtual machine.

After importing the data file, the system was briefly tested and worked flawlessly in all its three operating modes(Search, Reverse Geocoding, Address Lookup).

Routing Service

Since this service would be the core of our system, it was the only service that we would not be able to be acquire from a third party. From our preliminary findings, only routing services based on HM were suitable to support our system. GOS was simply not efficient enough, since they did their search in a full graph instead of an highly contracted sub-graph resulting, on average, in more nodes visited.

The immediate consequence of this finding was the fact that, since the pre- calculation step was a very time consuming operation, we would not be able to update our routing service in real time.

After selecting HM as our preferred search method it became clear that OSRM and GraphHopper were our prime candidates.

OSRM was installed twice: one using the source code and other using a Docker container. As expected, the Docker version was preferred in terms of installation was completely automated. After installation, we contracted and imported an OSM export file containing the map of Portugal using a generic profile. Finally, we ran the server with the resulting compiled map file and it performed flawlessly.

OSRM had an additional feature of being able to run using files contained in shared memory. This was highly desirable since it allowed us to substitute the compiled map file without having to stop the service. GraphHopper did not support this feature.

GraphHopper was simply downloaded and executed using Java Runtime Environment. It also took several minutes to contract the OSM export file containing the map of Portugal into a file containing a graph. After that the service was available to answer HyperText Markup Transfer Protocol (HTTP) protocol requests at port 8989. GraphHopper was tested using both the flexibility (GOS) and fast (HM) modes. As expected, the fast mode performed considerably faster.

Finally, we tested the GraphHopper Traffic Data Integration Demonstration using a map of the city of Cologne in Germany and real time data from sensors deployed locally and made available in real time via an HTTP service. This demonstration used GraphHopper in flexibility mode and real time data to change the speed from elements where the sensors were located. This allowed the demonstration to propagate regularly updated traffic conditions into the routing engine. Since sensors were only deployed in a few of the city's main roads, there was no real way of extensively testing this demonstration. Nevertheless it demonstrated the feasibility of such a system.

From the initial tests both routing services were very similar. Further automated tests would be run to compare the performance of both services. Nevertheless, the fact that OSRM could run using files contained in shared memory made it more promising.

Libraries for Web Map Applications

These libraries are better described in section 4.5.11.

During the preliminary work phase both Leaflet[Aga15] and Open Layers 3[Fou15] were tested. Since our initial implementation was very simple, only containing a slippy map and a simple form in which to set the source and target locations, both libraries performed adequately. Nevertheless, Leaflet was much simpler to use and the library file was many times smaller in size than Open Layers 3.

Since our Web browser based user Interface would be minimal, we chose Leaflet over Open Layers 3.

4.2.5 Manipulating Map Information

Manipulating map information, instead of building our own services, had several advantages. It was a much less intrusive approach that would allow us to reach our goals using third party components and guaranty a modular architecture that could be quickly improved and modified.

It would necessary to be able transform the OSM export files into a format that could be easily manipulated. That could be achieved by importing the OSM file into a PostGIS database using a tool like Osmosis.

After importing the map information into the PostGIS database, we were able to study how the elements that constituted the OSM data model were organized into the relational model of the database. We were also able to figure out how the Tags system was organized and how we could manipulate and transform elements and tags in order produce the necessary changes that can mimic traffic and air quality conditions.

Finally, using the same Osmosis tool, we were able to export the database back into an OSM export file that could be used by the many services that constituted the system (WMS, Routing and Geocoding).

At this point this strategy seemed to be the most promising technique to introduce the changes from the sensors into the services. Nevertheless it was still a very time and computationally heavy operation. If the map got any bigger, we would not be able to do it on the fly, specially if we were not using a dedicated machine to contain the database.

4.2.6 Conclusions

After the preliminary study of algorithms and solutions several preconditions were able to be enumerated:

1. Altering a routing engine would not be attempted;
2. Routing engines that used HM produced faster results and smaller workload at the cost of a precompiling step and the inability to change data in real time;
3. Since we would be using maps that had to be precompiled and this operation could take several minutes to complete, depending on the size of the map and the type of

machine, the system would not operate in real time. Map files that fed the services would only have to be updated regularly;

4. We would have to manipulate the map data fed to the services in order to be able to take into account the information that came from the sensors when calculating routes;

5. Manipulating tags like ”maxspeed”, or the Restrictions group was, at this point, the easiest way to manipulate map data;

6. Exporting the database and compiling the map regularly was a far from ideal strategy as we would struggle with larger maps. Efforts would have to be made in order to find a better solution;

7. OSRM posed as the most promising Web Routing service since it could use shared memory to update the data file without having to restart the whole system;

8. There was no real need to deploy the WMS and Geocoding services at this point. Nevertheless, if we were to scale up the system, this would eventually be necessary;

9. For its simplicity and reduced file size, the Leaflet library was preferable in order to implement the simple web browser based GUI necessary for the system.

These preconditions gave us a good vision of how an eventual system should operate and what it should look like. Taking them into account, we proceeded to elicit functional requirements by creating User Stories.

4.3 Requirements

The preconditions enumerated in the last section gave us a vision of what our system should achieve. This section will describe the process that started with functional requirements and ended with an initial architecture for our project.

From the user standpoint, our system was not much different from a traditional browser-based web map client. The traffic data integration was performed without any participation of the user. Therefore, functional requirements elicitation for our system was straightforward. Then we thought about the operational parameters that our system should obey to. With these parameters, we obtained restriction and quality requirements for our prototype.

These quality requirements did not tell us much about a possible system. We proceeded by further refining them and constructing scenarios that could better expose our operational parameters. These scenarios allowed us to decide which quality requirements were relevant to our architecture.

Knowing which quality requirements were architecturally relevant allowed us to choose architectural styles better fit our system. With this information, we devised an initial architecture and studied technologies and tools that helped us to construct our system.

4.3.1 Functional Requirements

Taking into account lessons learned from the preliminary work phase, we proceeded by constructing user stories that could capture how our browser-based client should operate. For each user story, we enumerated a series of acceptance criteria. If our system met all acceptance criteria, a certain user story should be accepted. Since these user stories would take too much space in the final report we opted to put them in Annex B.

Having created all user stories, we knew we would not be able to get them all accepted in the amount of time we had. Therefore we used the MoSCoW prioritization to determine which user stories we should observe (Must Have) and which stories we should leave for later. Next, we will present the user story prioritization.

User Stories Prioritization

In a project with fixed time, it is often not possible to attain all proposed requisites. As such it becomes of the utmost importance to prioritize Requirements / User Stories [Con16a].

MoSCoW is a prioritization technique used for helping understanding and managing priorities. This method categorizes the User stories into four relative priorities:

- **Must Have:**
- **Should Have:**
- **Could Have:**
- **Won't Have this time:**

The use of these categories clearly indicates the importance of each item and the expectations for its completion. Next, we will present all the elicited user stories and its relative importance for the success of our system:

User Stories - End User

ID	Description	Priority
US-1a-1	As an End-User, I want to manipulate the map zoom and visualization window so I can better visualize the desired map elements.	Must-Have
US-1a-2	As an End-User, I want to be able to visualize traffic density so I can better assess the present traffic situation.	Must-Have
US-1a-3	As an End-User, I want to be able to visualize air quality so I can better assess the present environmental situation.	Could-Have
US-1a-4	As an End-User, I want to be able to visualize parking density so I can better assess the present parking availability situation.	Could-Have
US-1b-1	As an End-User I want be able to swap between two GUI modes (Search and Directions) so I can better manipulate the interface with different behaviors and functionalities.	Must-Have
US-1bi-1	As an End-User I want be able to select a new point on the map each time I press the mouse button.	Must-Have
US-1biA-1	As an End-User I want to translate any set of valid geographic coordinates (Latitude and Longitude), inserted in a valid form, into a valid point on the map so I can find any geographic location.	Must-Have
US-1biA-2	As an End-User I want to translate any set of valid geographic coordinates (Latitude and Longitude), inserted in a valid form, into a valid street address if one exists so I can find any address from a given set of coordinates corresponding to it.	Must-Have
US-1biA-3	As an End-User I want to translate any valid address, inserted in a valid form, into a valid point on the map so I can find any element on the map.	Must-Have

US-1biA-4	As an End-User I want to select with the pointing device any point on the map and translate it into a valid set of geographic coordinates and a valid address if one exists so I can retrieve information about any point on the map without knowing its address or geographic coordinates.	Must-Have
US-1bii-1	As an End-User I want to be able to select a source and a target on the map so I can obtain directions from the source to the target.	Must-Have
US-1bii-2	As an End-User I want to be able to insert valid addresses into a source and a target valid form so I can obtain the fastest path from the source to the target.	Must-Have
US-1bii-3	As an End-User I want to be able to insert valid geographic coordinates into a source and a target valid form so I can obtain the fastest path from the source to the target.	Must-Have
US-1bii-4	As an End-User I want to be able to choose from several mobility profiles so I can choose which mean of transportation I will use to reach my target.	Should-Have
US-1bii-5	As an End-User I want to be able receive directions from source to target after my route has been plotted so I can better navigate the route.	Must-Have
US-1bii-6	As an End-User I want to be able choose from several routing types so I can better plan my trip.	Could-Have
US-1biiA-1	As an End-User I want to translate any set of valid geographic coordinates (Latitude and Longitude), inserted in a valid form, into a valid point on the map so I can find any geographic location.	Must-Have
US-1biiA-2	As an End-User I want to translate any set of valid geographic coordinates (Latitude and Longitude), inserted in a valid form, into a valid street address if one exists so I can find any address from a given set of coordinates corresponding to it.	Must-Have
US-1biiA-3	As an End-User I want to translate any valid address, inserted in a valid form, into a valid point on the map so I can find any element on the map.	Must-Have

US-1biiA-4	As an End-User I want to select with the pointing device any point on the map and translate it into a valid set of geographic coordinates and a valid address if one exists so I can retrieve information about any point on the map without knowing its address or geographic coordinates.	Must-Have

Table 4.1: User Stories Prioritization -End User

User Stories - Unauthenticated System Operator and Super User GUI

ID	Description	Priority
US-2a-1	As an Unauthenticated System Operator or Super User I want to be able to log into the system so I can log in and access all system functionalities.	Must-Have

Table 4.2: User Stories Prioritization - Unauthenticated System Operator and Super User GUI

User Stories - Authenticated Super User GUI

ID	Description	Priority
US-3a-1	As a System Operator Entity I want to be able to visualize traffic history so I can better assess the evolution of the traffic and better plan transportation policy.	Must-Have
US-3a-2	As a Super User I want to be able to manage authorizations so I can better assess and manage system operators.	Must-Have
US-3a-3	As a Super User I want to be able to manage database tables so I can better assess and manage stored information.	Must-Have

US-3a-4	As a Super User I want to be able to manage Celery Worker tasks so I can better assess and manage operation.	Must-Have
US-3a-5	As a Super User I want to be able to manage Celery Beat periodic tasks so I can better assess and manage operation schedule.	Must-Have
US-3a-6	As a Super User I want to be able to manage Celery Workers so I can better assess and manage worker operation.	Must-Have
US-3a-7	As a Super User I want to be able to visualize task history so I can better assess whether or not the system is functioning correctly.	Must-Have
US-3a-8	As a Super User I want to be able to Log out so I can successfully terminate my session and limitate access to unauthorized people from any machine using his account.	Must-Have

Table 4.3: User Stories Prioritization - Super User

User Stories - Authenticated System Operator GUI

ID	Description	Priority
US-4a-1	As a System Operator I want to be able to manage database tables so I can better assessand manage stored information.	Must-Have
US-4a-2	As a System Operator I want to be able to manage Celery Worker tasks so I can betterassess and manage operation.	Must-Have

US-4a-3	As a System Operator I want to be able to manage Celery Beat periodic tasks so I canbetter assess and manage operation schedule.	Must-Have
US-4a-4	As a System Operator I want to be able to manage Celery Workers so I can better assessand manage worker operation.	Must-Have
US-4a-5	As a System Operator I want to be able to visualize task history so I can better assesswhether or not the system is functioning correctly.	Must-Have
US-4a-6	As a System Operator I want to be able to Log out so I can successfully terminate my session and limitate access to unauthorized people from any machine using his account.	Must-Have

Table 4.4: User Stories Prioritization - System Operator

4.3.2 Constraints

After eliciting functional requirements, we obtained the projects Constraints. These referred to restrictions in the software design that we should follow for the project to be successful. While functional and nonfunctional requirements could be hierarchized, flexed or dropped, with the stakeholder's agreement, to reach a consistent architecture, constraints should be met to reach the minimum threshold of success.

For our project we elicited one major constraint: All software and services used shall be completely free of any charge;

4.3.3 Non-Functional Requirements

Non-Functional Requirements referred to minimum quality attributes the system should observe to operate properly. These requirements were independent of any specific behavior defined by the functional requirements/ user stories.

For our small prototype, we defined eight non-functional requirements:

Extendability: The System shall be able to be grown by adding new features, functionalities, and products;

Interoperability: The System shall be able to be integrated with other systems without any modification;

Modularity: The system shall allow the installation of new components and the modification of existing ones without any changes to other components;

Portability: The system shall be able to operate on all the major operating systems;

Reusability: The modules and components developed for this system shall be able to be used in other systems;

Scalability: The system shall be able to scale horizontally;

Security: The system shall be able secure and resist attempts from nonauthorized users to use/ tamper data or services while providing access to legitimate users;

Usability: the system shall be easy to understand, to learn and to use.

4.4 Initial High Level Architecture

4.4.1 Architectural Drivers

The quality requirements elicited in the last section did not give us much information. To further refine these requirements, we built scenarios to understand how they affected our system's operation. Then we organized scenarios into a utility tree and prioritized them. The utility tree can be consulted in Annex C.

By analyzing the utility tree, we determined which quality requirements were architecturally significant and should drive the architectural design.

4.4.2 Architecturally Significant Requirements

This section describes how the elicited nonfunctional requirements affected our architecture. From the original 8, we selected 6 architecturally significant requirements. We will describe how each one influenced our initial architecture:

Extendability: we guaranteed this requirement by developing completely documented and understood exchange and file formats. We used well-defined communication protocols;

Interoperability: like the requirement above, we guaranteed this requirement by developing completely documented and understood exchange and file formats. We used well-defined communication protocols;

Modularity: was guaranteed by encapsulating all components and only allowing them to communicate using a well-defined interface. Since these components were black boxes, we could freely trade them as long as the interface was the same;

Portability: was achieved either by developing the code in a high-level computer language whose compiler supported all major platforms or by using some kind of virtual machine or interpreter abstraction between our system and the operating system;

Reusability: we guaranteed this requirement by developing completely documented and understood exchange and file formats. We used well-defined communication protocols;

Scalability: if the workload became too great, the system would split work split between several machines deployed according to its needs. We could achieve this using a load balancer or a distributed task queue.

4.4.3 Architectural Style

After analyzing the architecturally significant requirements, we had a good idea how our system should operate. We knew our system should be composed of several subsystems. Part of the system should process sensor data and the other would serve a web browser-based client. Since this subsystem had very different behaviors, two different architectural styles were selected.

Pipe and Filter Architectural Style By analyzing how our system should work, we came to the conclusion that these complicated processes, like processing sensor data or compiling map data, could be decomposed into several smaller tasks. Events, like new sensor data arriving, triggered a sequence of steps where data flowed unidirectionally and got transformed into useful information. This information was stored in a database, a memory cache or a hard disk.

Therefore, a Pipe and Filter Architectural style [HW04] made sense as it depicted the exact behavior described for this part of our system. This architectural style also allowed us to incorporate the most architecturally significant requirements:

- **Extendability:** the system could be easily extended by adding extra components to the pipe. Since we were using well-documented interfaces, exchange and file formats and well-known communication protocols, this could be easily achieved;
- **Interoperability:** any existing or future system could be linked to either end of our system pipeline as its interfaces, exchanges and file formats were well documented and used well-defined communication protocols;
- **Modularity:** since every process in our system acted as a black box to the outside world and had well-defined interfaces that communicated through well-known protocols, it would be easy to swap any component of our system without having to modify the rest of the system;
- **Portability:** this quality requirement was out of the scope of this architectural style. A lower abstraction layer would deal with it;
- **Reusability:** any component or module could be used in other current or future systems since it had well-documented interfaces, exchange, and file formats and it communicated through well-known communication protocols;

- **Horizontal Scalability:** this architectural style made it easy to identify bottlenecks in our system. To resolve these bottlenecks, we would only have to add components parallel to the bottleneck and a load distributor to split load between the two components.

Service Oriented Architectural Style On the other hand, the part of the system that dealt with web browser clients had a strong emphasis on HTTP services replying to client requests. Therefore, this part of the project should emphasize this type of behavior by using a Service Oriented Architecture style.

This architectural style also allowed us to incorporate most architecturally significant requirements:

- **Extendability:** was achieved by adding new services to the system;
- **Interoperability:** any existing or future system could use existing services as its interfaces were well documented and used well-defined communication protocols;
- **Modularity:** since every service in our system acted has a black box to the outside world and had well defined interfaces that communicated through well known protocols it would be easy to swap any service without having to modify the rest of the system;
- **Reusability:** any service and module could be used in other current or future systems since it had well-documented interfaces, exchange, and file formats and it communicated through well-known communication protocols;
- **Portability:** this quality requirement was out of scope of this architectural style and had to be dealt with in a lower abstraction layer;
- **Horizontal Scalability:** this architectural style would be easy to scale horizontally. We could solve fluctuations in demand while maintaining quality of service by deploying additional instances of the same service and a load balancing mechanism that could distribute requests among the available service instances.

4.4.4 System Decomposition

After the preliminary work phase, we knew that our system could be split into of three different sub-systems:

1. An event based system that received data from sensors, extracted, transformed and loaded it into a database;
2. A time-based system that used a chronometer to schedule operations. This system should be responsible for dumping the database information into a format that could be used by the routing engine compiler and allow the routing engine contracted map to be updated;
3. A browser-based web client that made HTTP requests to several web services;

The first two systems had considerable similarities as information flowed unidirectionally and the flow was always started by an event that triggered a sequence of processing steps. The third system was based on a request-reply model. These systems could be further decomposed into several relatively simple tasks. In order to determine which tasks were needed, business process notation was used to specify the business processes that happened in each component.

The third system was very similar to the traditional web map routing implementation.

4.4.5 System High Level Architecture

After analyzing the Business Process Management Notation (BPMN) diagrams in Annex D, we designed a high level architecture. Figures 4.2 and 4.3 represent two of this architecture views.

These diagrams had several type of components. Only the Internal ones would have to be developed.

Distributed task queue: would be central to the application. Since it had already been established that the system would not operate in real time, the introduction of message queues allowed loose coupling between components. It could act has a message buffer allowing tasks to be processed in batches. If the system needed to scale horizontally, we could parallelize key operations by simply adding more workers without any further modifications to the system.

Workers: instead of having different components we could create tasks to be executed by workers. This tasks would allow better system management as any worker would be able to execute any tasks. This fact would facilitate horizontal scalability;

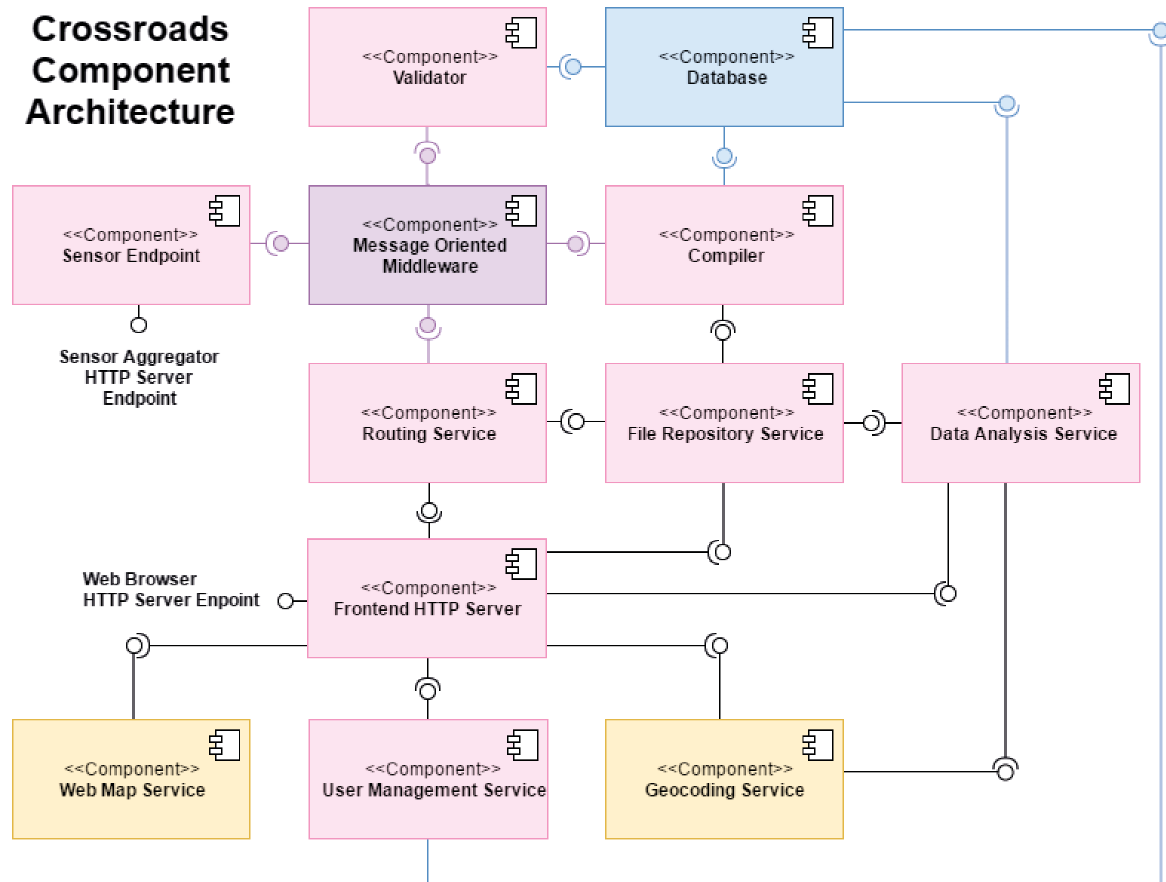
Tasks : At this point, we identified two different tasks. As the project evolved, we would try to split these tasks further into simpler subtasks. We will describe both tasks identified:

- **Validator:** Validated the message content, did the necessary requests to other services to fill missing information and inserted the reading into the database;
- **Compiler:** At the scheduled time it exported the database into a file or a cached data structure. It would then proceed with the necessary compilation step. It would finally store compiled map data in a file repository;

Internal components to be developed: We would also need to develop several internal components. We will briefly describe each of these components.

- **Sensor Endpoint:** was composed of an HTTP server that received messages from a sensor aggregator. After extracting the message payload, it tried to parse and classify it. If it was successful it would create the appropriate tasks and positions them in the right queue;

Figure 4.2: Crossroads Initial Top View Architecture



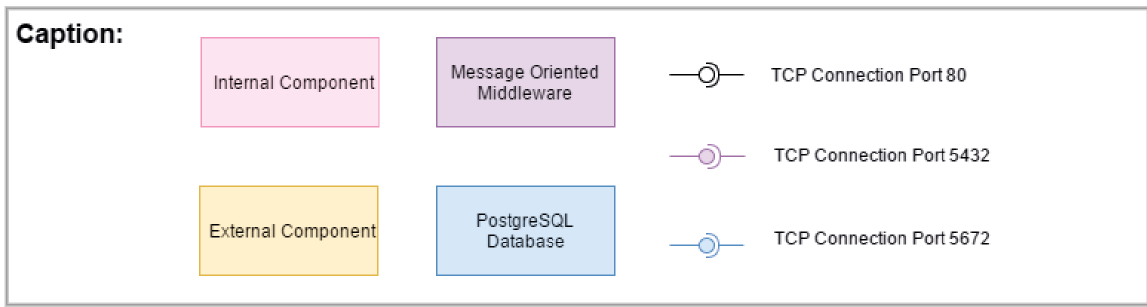


Figure 4.3: Crossroads Initial Component Architecture

- **File Repository Service:** acted as a centralized repository for all static files (HyperText Markup Language (HTML), JavaScript (JS), OSRM...);

- **Routing Service:** It was composed of a controller and the OSRM routing service itself. When it received a message that new OSRM files were available, it would downloads it from the file repository service and uploads it to the shared memory;

- **Data Analysis Service:** analyzed data contained in the database, queues, and workers and fed it to the System operator console;

- **User Management Service:** managed user registry and authentication;

- **Frontend HTTP Server** It would serve the web browser based client's requests and act as a relay between the client and the web services that fed the system.

Besides this internal components, the system would have a few external components like the WMS and the Geocoding Service that fell outside the scope of the project for now.

The **Sensor Aggregator** component was an abstraction. It could take many forms depending on what data we fed data to the system. We would create a sensor endpoint task that could uniformize data from each potential data source into a common data format that could be used by the system. Finally, the system would be supported by a relational database that could store relevant information.

4.4.6 Conclusions

At this point, we had a good idea of what components should compose our high-level architecture. This architecture was only meant to guide us through the early stages of development. It was probable that, as more information became available, our system would evolve.

Using this high-level architecture as our starting point, we continued the study of technologies that allowed us to develop our prototype and meet all operational requirements.

4.5 Technologies

After specifying a high-level architecture, we studied the necessary technologies to support it. These technologies allowed us to

This section aims at describing and comparing different technologies that were in our project. Since Ubiwhere had adopted a standard project template, many of these choices had already been made for us. Nevertheless, we described all the template's features and exposed the reasons why they were chosen over other possible options.

4.5.1 Yet Another Django Project Template

While developing a project, it took developers several to set up the proper system environment. Also, when maintaining or modifying older systems, the lack of a common project template made it harder for developers, who had often not been the ones to develop it, to accomplish the task at hand in the allocated time.

In an effort to mitigate these problems and standardize software development, the company had adopted Yet Another Django Project Template (YADPT) [Kha17] as its standard project template [Ubi17]. While there were several templates already available, YADPT took into account the company's necessities by combining key features:

- Fully automated template using Docker Containers [Inc16b];
- Automatic generation and renewal of Let's Encrypt [Gro17a] Secure Socket Layer (SSL) certificates to allow HyperText Markup Transfer Protocol Secure (HTTPS) protocol communication;
- Adherence to Django [Kha17] development best practices [Fou17a];
- Different environments for development, staging, and production provided.

This template featured a preconfigured docker-compose [Inc17a] environment that containers several Docker containers necessary for our system. We could add more containers to this environment by just editing the Ain't Markup Language (YAML) files. These components also could be easily swapped by others with minimal modifications to the project.

We will now describe the components already deployed:

- **PostGIS database [Fou16g]:** featuring a spatial and geographic extension for the PostgreSQL object relational database [Gro17b];
- **Redis in-memory data structure store [San17]:** could be used as a database, cache and message broker. In this particular template, Redis was being used by Django has a cache feature;
- **Django Based Web Application:** featured an empty Django Project, different requirements and settings files for development, staging, and production. The container also feature the Gunicorn Web Server Gateway Interface (WSGI) [Che17];
- **Nginx Web Server [Inc17b] :** web server/ reverse proxy to our application server;
- **Certbot [Fou17b]:** to automatically create and renew Let's Encrypt [Fou17g] SSL certificates.

Since the company had adopted this template, we aligned our technology choices with it to facilitate further development of our prototype, its reusability, and maintainability.

4.5.2 Programming Language

Although a good portion of our prototype would be constructed taking advantage of already available services, we still needed to develop several components.

When choosing a programming language, we took three factors into account:

- The language selected possessed the necessary frameworks and libraries that should allow the system to meet operational requirements;
- The developed components were compatible with the existing project template;
- The language selected was compatible with the intern's skill set.

Since we would develop a series of small prototypes in rapid iterations, we should select a programming language that could allow fast development cycles, was robust enough to support web services and job automation.

Taking into account these two options were considered:

- **Python:** was an high-level, general-purpose, interpreted programming language that emphasized code readability and syntax being considerably less verbose than other languages like Java. Since it was a interpreted language, it was well suited for the development of small components, modules, and micro-services as well as fast prototyping. Python had the advantage of being the language used by the Django-based Web Application already present in the template [Fou17e];
- **Java:** was a high-level, general-purpose, object-oriented language intended to be run on a Java Virtual Machine (JVM). By using a JVM, the developed code was able to run on any machine, regardless of its computer architecture as long as there was an implemented JVM [Ora17].

When taking into account the factors enumerated above, Python was clearly a more suitable candidate. Even though it should be relatively easy to swap the web application container for another that could support a Java based web framework, we would not be taking advantage of all the knowledge already amassed at the company and it would be hard to promote reusability of our code.

4.5.3 Django Web Framework

After selecting Python as our programming language, we needed to select a Web Framework to support our server. The first option we had to take was whether or not we wanted a full stack framework. Since our system would have databases and Message Oriented Middleware (MOM) included, not using a full stack framework would mean we would have to install all needed libraries by ourselves.

Since we wanted to spend as little time as possible developing the necessary infrastructure to support our system, we ended up going for what was already implemented in the MOM template and selecting Django [Fou17a].

Django is a full-stack Python web framework that specialized in building web applications using as little code as possible. It comes with the necessary features to deal with the most common Web development task like authentication, content administration, testing and much more straight out of the box. Every Django project is split into one or more apps to promote modularity and reusability. These apps can be reused and obtained from repositories. This allows developers to create rather complex web applications without developing every component from scratch.

Since Control is already handled by the framework,it follows a Model View Template Architectural Pattern (MVT):

- **Model:** constitutes an abstraction of the data layer. By manipulating model instances, we can create, access, manipulate, delete and validate data objects. These objects are replicated in the relational database present bellow this layer;

- **View:** constitutes the business logic of our application. Using them, we can manipulate models and extract the necessary information to create dynamic web pages using templates;

- **Template:** represents the presentation layer. Each Django application contains the necessary templates that use information generated by the business logic layer to generate dynamic web pages.

4.5.4 Message Broker

While developing our initial architecture, we had already come to the conclusion that our prototype should be a distributed system sending and receiving messages from very heterogeneous platforms.

To achieve such goal, it would be necessary to create a communication layer that could isolate the various components, guaranty loose coupling, allow our system to function asynchronously, and assure reliable communication without any modifications to the already selected components. These features could be achieved using a message broker.

We analyzed two of the most common message broker software options:

- **RabbitMQ:** was an open source message broker that used the Advanced Message Queueing Protocol (AMPQ) protocol allowing asynchronous messaging and component decoupling. It was reliable and easy to use requiring minimum configuration [Pre07];

- **Redis:** “is an open source (BSD licensed), in-memory data structure store, used as a database, cache and message broker. It supports data structures such as strings, hashes, lists, sets, sorted sets with range queries, bitmaps, hyper logs and geospatial indexes with radius queries. Redis has built-in replication, Lua scripting, Least Recently Used (LRU) eviction, transactions and different levels of on-disk persistence, and provides high availability via Redis Sentinel and automatic partitioning with Redis Cluster” [San17].

Both message brokers were relatively straight forward to use with Django, since there were integrated libraries and minimal configuration would be necessary. We chose Redis because it was already present in YADPT as its web application cache mechanism.

4.5.5 Distributed Task Queue

Task queues were mechanisms that allowed the distribution of tasks across worker threads or processes. Produced tasks were created and stored in queues inside a message broker. Once a new task became available, the worker would perform it. A hypothetical system could consist of a variable number of workers and queues, allowing high availability and horizontal scalability. These tasks could either be produced by events like incoming messages or be scheduled to be performed.

The most commonly used distributed task queue Python library was Celery [Sc17], an open source asynchronous task queue based on distributed message passing. There were other options for simpler projects like Redis Queue [Dri17] or Huey [Lei17], nevertheless they were not as complete as Celery. Since Django supported Celery out of the box, we could implement and call our tasks directly from our apps and Django will deal with all the underlying complexity.

Using the proper libraries we could deploy celery in several modes:

- **Celery worker:** created a worker process that monitored the queues for tasks to perform. Once a new task arrived, it would get delivered to the worker that would execute it and reenter a holding pattern;

- **Celery beat:** acted as a scheduler that created tasks to be executed by workers at regular intervals or a given time/date;

- **Celery cam:** was a system monitor that allowed task and workers and the general state of the distributed task queue system to be monitored.

4.5.6 Relational Databases

Web Map Services required large quantities of information to run properly. To store and query so much data, a relational database was normally used. The the OSM tools used the PostgreSQL [Fou16g] open source object-relational database and the PostGIS [Fou16g] spatial database extender as its default database.

Since a container featuring PostGIS already came in the YADPT template, there was not much point choosing another relational database.

PostGIS: was a spatial database extender for the PostgreSQL database. It supported geographic objects and allowed spatial data and location queries to be more easily handled in SQL. This database extender was necessary to properly manipulate our OSM map data and power tools like Mapnik map renderer or the Nominatim Service.

4.5.7 Automatic Deployment Tools

Deploying services by hand was a very time-consuming process, being able to automate this process was a necessity. Creating self-sufficient software containers, able to be replicated, built and deployed as needed, greatly contributed to the system scalability, modularity and fault tolerance.

Using a deployment tool should also guaranty the portability quality requirement since the tool acted as an abstraction layer between the system and the operating system.

Since we were using the YADPT template that was based in Docker [Inc16a], there was not much point in investigating other potential alternatives.

Docker: was an open source tool that automated software deployment inside containers. "Docker containers wrap a piece of software in a complete filesystem that contains everything needed to run: code, runtime, system tools, system libraries – anything that can be installed on a server. This guarantees that the software will always run the same, regardless of its environment" [Inc16b].

By isolating the running application from the rest of the infrastructure, this container allowed for software to run on all major Linux Distributions and Microsoft Windows while providing an extra layer of protection for the application guarantying the system's portability.

This container also contributed to system's modularity as they could be swapped without the need to modify the system acting as a black box with a well-known interface.

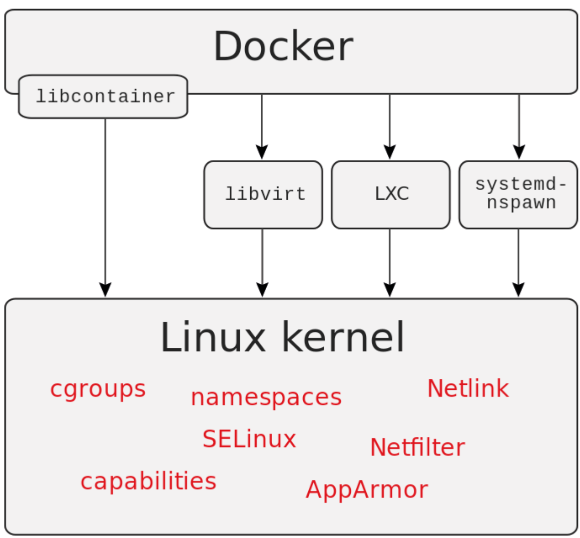


Figure 4.4: Docker - Linux kernel interface architecture

Docker was able to access resource isolation features from the OS kernel allowing several Docker containers to run within the same machine and eliminating the necessity of starting and maintaining several virtual machines. Since a single server or virtual machine was able to run several containers at the same time, this feature created the necessity of being able to define how these containers related and communicated among themselves allowing more complex systems to be built.

Docker Compose [Inc17a]: was a tool used to define, build and deploy multi-container applications. By using a compose YAML file, we were able to configure all the application's services and how they should interact with each other.

4.5.8 Web Server Gateway Interface

The built-in server that came with the Django Web Framework was only designed for development purposes. To deploy our application in a production environment, it would be necessary to deploy a more robust alternative like Apache [Fou17f] or NGINX [Inc17c].

To use a more robust web server, it was necessary to create an interface between the web server and the application. The WSGI [Eby10] specified by PEP 3333 was created such interface. The YADPT template, in its production environment, contained the Gunicorn Python WSGI HTTP Server [Che17].

4.5.9 Web Servers

As stated earlier, it was necessary to deploy a more robust Web Server in production environments. Since we already had Gunicorn WSGI to deal with the dynamic contents generated by Django, we would need a Web server that could act as a reverse proxy and should excel in serving static contents and concurrency.

The two most popular options were:

- **Apache:** was a robust, commercial grade, freely available HTTP Web Server. Although it was more versatile, it had limitations regarding replying to a great number concurrent requests at the same time;
- **NGINX:** was a HTTP Web Server that focused on concurrency handling and serving static contents while passing dynamic requests to the application server.

In our particular NGINX was a much more attractive option. It could serve a lot of requests for JS and Cascade Style Sheet (CSS) and other static resources, while acting as a reverse-proxy to the dynamic contents generated by our application server. The fact that glsyadpt already featured this Web HTTP server there further supports our choice.

4.5.10 Let's Encrypt and Certbot

Passing unencrypted information through the Internet was a security risk. When a client visited a certain website he had no guaranty whatsoever that the service at a certain domain was legitimate. To solve these issues, it was necessary for entities to get certified by a certificate authority. Using a certificate, services could identify themselves to their clients and be able to negotiate a session key, enabling HTTPS communication using the SSL secure protocol.

YADPT production environment features a Certbot [Fou17b] container that automatically create and renews Let's Encrypt[Gro17a] certificates to allow HTTPS protocol communication using our HTTP web server.

4.5.11 Libraries for Web Mapping Applications

There were several JS libraries that allowed us to create web maps supported by a web page:

Leaflet [Aga15]: "Leaflet is an open-source JS Library for mobile-friendly interactive maps". It allows for the creation of web pages featuring tiled web maps with optional tiled overlays.

It was designed to be efficient across all major desktop and mobile platforms and focus on simplicity, performance, and usability. Although it was very simple at its core, it could be extended with external plug-ins in order to accommodate more advanced features.

Leaflet Routing Machine [Lie15]: was a JS library that supported multiple routing machine backends like OSRM or GraphHopper. It integrated perfectly with Leaflet main library and supported routing from source to target with the possibility of via points and the use of waypoints. It also supported multiple languages.

Leaflet Control Geocoder [Lie16]: was a JS library that acted as a simple geocoder and supports multiple geocoder backends like Nominatim, Google Geocoding API or Mapzen. It integrated with Leaflet and allowed address and geographic coordinates translation.

OpenLayers 3 [Fou15]: "OpenLayers makes it easy to put a dynamic map in any web page. It can display map tiles, vector data and markers loaded from any source. OpenLayers has been developed to further the use of geographic information of all kinds. It is completely free, Open Source JS, released under the 2-clause Berkeley Software Distribution (BSD) License (also known as the FreeBSD)." This JS library contains much more features than the Leaflet library at the cost of added complexity and a much larger script to download.

4.6 Conclusions

During this chapter, we began by deploying and testing possible tools and services that could integrate a possible prototype. With the knowledge gathered from this preliminary work phase, we elicited the project's functional requirements. Then we chose and refined, through a series of scenarios, our quality requirements. Since these scenarios gave us an idea of how our quality requirements influenced our architecture, we specified an initial architecture for our prototype nd studied technologies that could support it.

The next chapter will describe the development phase which consisted of four iterations and resulted in a final prototype to be delivered to Ubiwhere.

Chapter 5

Development

5.1 Introduction

In the last chapter, we began by deploying services and tool discovered so far. This activity gave us a better understanding of how we should develop a web routing system that took into account traffic information when calculating routes. Then we proceeded to write and prioritize user stories that would function as functional requirements and elicited restrictions and quality requirements for the system. To refine these quality requirements, we wrote scenarios that allowed us to determine and prioritize architecturally significant requirements and find two architectural styles (Pipe and Filter and Service Oriented architecture) that could comply with them. Finally, we reached an initial architecture and studied technologies for development.

This chapter will describe the iterative development process that took part during most of the second semester and resulted in a series of small experiments and prototypes.

During the development phase, we took four monthly iterations:

- **Iteration One:** we started by risk discovery and elaborated a risk mitigation plan. This plan dictated that we should start by selecting an adequate data source for our project. Since we did not have the tools necessary to make this evaluation, we developed a browser based web map client and the necessary services to analyze sensor coverage. We took this opportunity and developed a client that could support all must have user stories. This client allowed us to evaluate and select an adequate data source;

- **Iteration Two:** we started by assessing the possibility of exporting the whole map database on the fly and compiling to a format that could be used by OSRM. Since this would not be feasible, we found the OSRM experimental traffic feature as the alternative. Then we assessed risks and successfully evaluated this experimental feature to mitigate them;

- **Iteration Three:** this iteration had the objective of constructing a complete prototype and evaluate it. At this point, we did not know whether or not the edges that were being fed to the routing engine using the OSRM traffic feature were present in the graph map. This doubt was a risk that should be mitigated. After some research, we found OSRM match service to reverse geocode our sensor paths. Since

this service used the same graph map as our routing engine, we could be sure that the edges produce would be on the map. This service also simplified our prototype. After constructing it, we evaluated it against the GraphHopper traffic integration demonstration that used the same data source with partial success;

- **Iteration Four:** Even though we had developed a fully functional prototype during the last iteration, it did not meet operational requirements. During this iteration, We developed a more robust system that was be thoroughly tested and accepted by the stakeholders.

We chose to dedicate a whole section to testing. Django had an automated testing feature, we were able to perform unit and integration tests without having to rely on other tools. After developing more than one hundred tests, we deployed Django's built-in application server to automate test execution.

After successfully performing the automated deploying the system using Docker-Compose, we further tested our system to assess whether or not the browser-based web map client was working properly in different browsers and the system was properly serving client's requests. Then we performed usability tests. We asked 5 people, unrelated to the project, to perform several tasks using our browser-based map client. Finally, we tested if our system could meet with functional and functional requirements to be accepted by the stakeholders.

5.2 Iteration One

5.2.1 Planning and Risk Assessment

After defining requirements and a possible high-level architecture, we proceeded to development. To better understand the tasks at hand, we identified risks to the project, prioritized them and dealt with the more urgent threats.

From the start of the project, the lack of a complete dataset had been considered the greatest risk for this project. This risk would have to be dealt right away before we could proceed.The full iteration one risk analysis document is available in appendix E.2.

5.2.2 Risk Mitigation Activities

From the initial steps, we knew there were several partial data sources available. Evaluating whether or not these sources were adequate for our project would require tools to be developed. To better visualize our data sources we decided to develop a web browser based client and the necessary services to support it. The client layered traffic data on a web map and allowed us to assess map coverage and the quality of received data.

This initial system, shown in figure 5.1, had a very simple architecture.

Three components were developed in this initial step:

- **Sensor Endpoint:** A very simple python script that retrieved traffic data from its source, processed it into a common format that could be parsed by the frontend

server, sent the produced information to the frontend server using an HTTP POST request and schedule its next iteration. This script was set to run every ten minutes. There would have to be a different sensor endpoint for each potential data source;

- **Frontend Server:** developed using the Django web framework, it had three functions:

  1. Process and store information received from the Sensor Endpoint into a database;
  2. Relay all clients geocoding, web map, and routing requests to the appropriate external demonstration servers;
  3. Build the Geographic JavaScript Object Notation (GeoJSON) object that represents the sensor network using data stored in the database;
  4. Reply to client's map layer requests with this GeoJSON object.

- **Web Browser Client Page:** developed using the Leaflet Library extended with some extra functionalities. It supported all Must Have User Stories. Since our data source was updated regularly, we used the jQuery [Fou17i] library to support asynchronous HTTP (Asynchronous JavaScript And XML (AJAX)) requests [Fon17], to allow our page to poll the server at regular intervals and update the traffic situation presented on the screen.

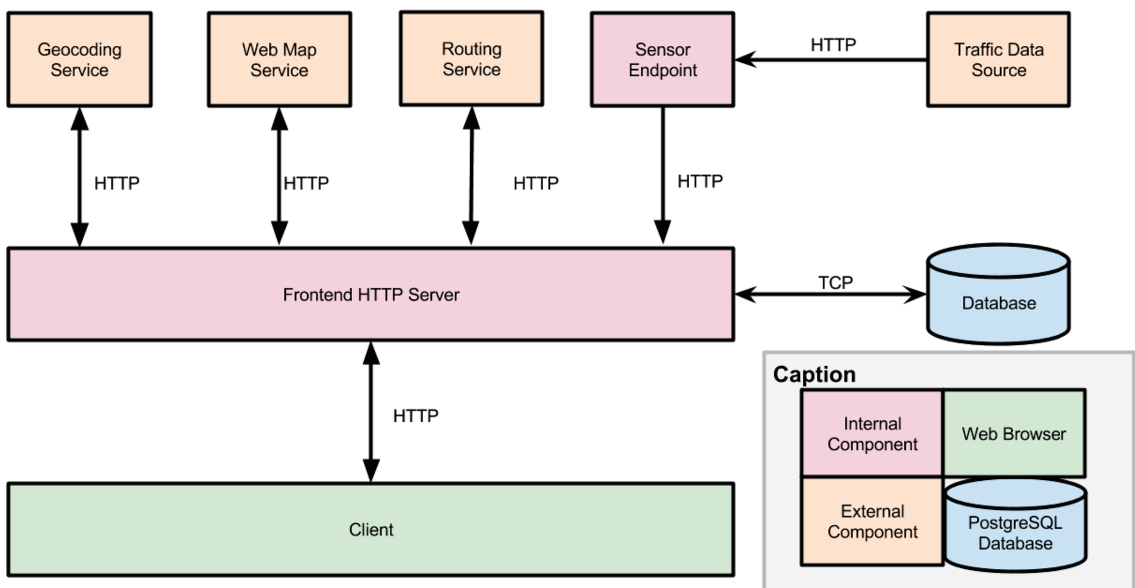


Figure 5.1: Iteration one System Architecture

While developing this initial browser-based client, we took the opportunity to develop all Must Have User Stories for this initial interface. Even though the fastest routes layered on the browser-based map interface did not take into account traffic data, this client was similar to the final prototype's client.

5.2.3 Conclusions

This initial system allowed us to visualize sensor coverage and data quality. We were able to analyze how the client used the final solution.

After analyzing data sources, we chose the Cologne traffic data source. This source presented data from more than one hundred sensor sources that were updated every five to ten minutes. It had the added advantage that a demonstration that used GraphHopper Routing Engine in compatibility mode to simulate the effects of traffic in a routing engine was already available. Later, this demonstration proved helpful to evaluate our initial prototype. The figure 5.2 shows the Cologne data source layered on the web map in our web browser based client.

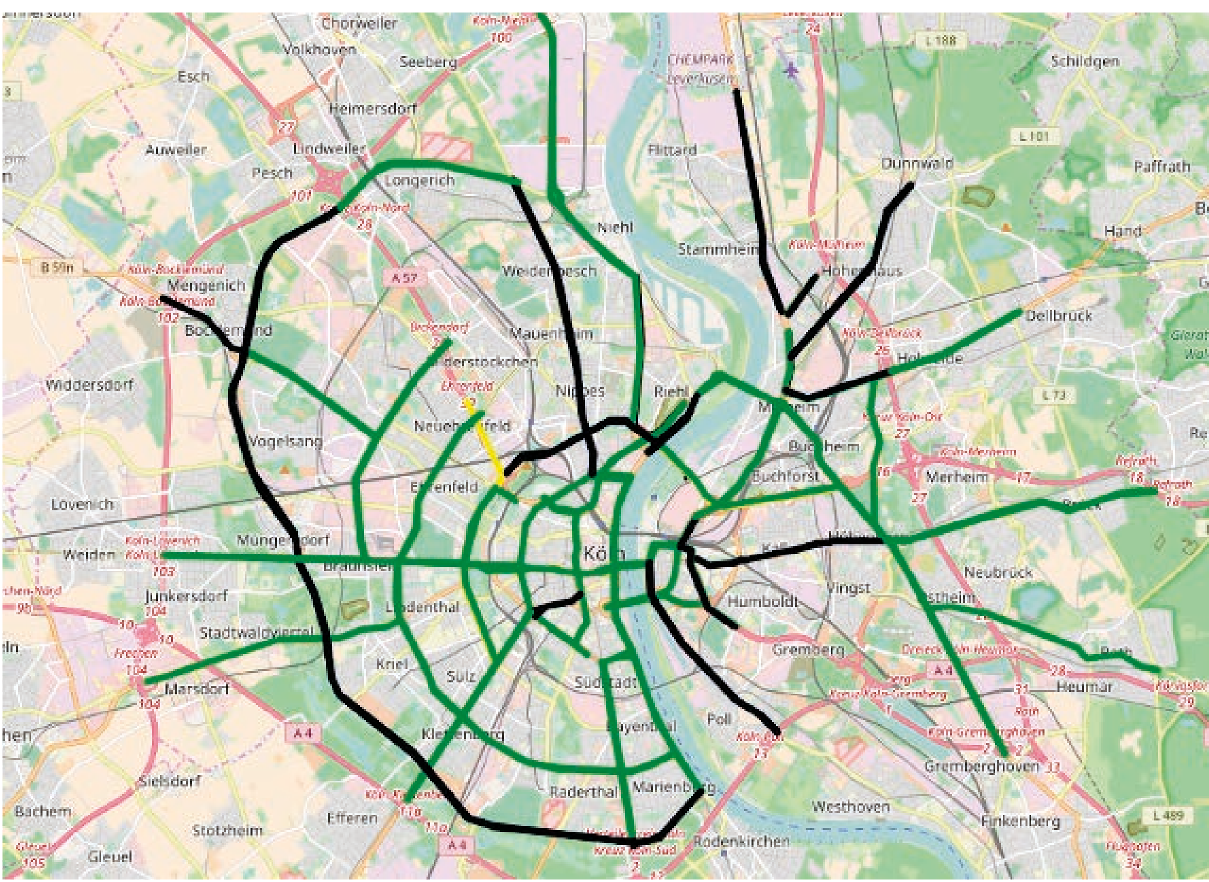


Figure 5.2: Cologne sensor data source layered on the map

Having chosen an adequate traffic data source and developed a browser-based web map client, we proceeded to the second iteration.

5.3 Iteration Two

5.3.1 Planning and Risk Assessment

During the last iteration, we had chosen an adequate dataset and constructed an initial prototype that implemented all must have user stories. At this point, we had to assess

whether or not our initial architecture was appropriate for this data source.

One of the major concerns regarding the original architecture was the amount of time and processing effort needed to export the PostGIS database containing updated map data into the *osm.pbf* export format. The original Cologne region map export file had 160 MB. Exporting the PostGIS database, extracting and contacting the map update the OSRM service took several minutes. This procedure would be too time and resource consuming for our prototype. We had to find a different method of integrating traffic information into the routing engine.

After some research, we learned that OSRM, since version 4.9.0., featured experimental traffic support. To be able to integrate traffic data into the OSRM map contraction step, we had to generate a Comma-Separated Values (CSV) file containing a graph edge (represented by a source and a target node *osm_id*) and a speed value in each line.

Even though this feature was still experimental, it had significant advantages over having to export the complete PostGIS database. We were able to manipulate edges that constituted the contracted map graph, considerably simplifying our prototype. Since we only had to store Nodes, Edges and Sensor Readings in our database, the full PostGIS database featuring a complete OSM map was no longer necessary.

Before making any changes, we assessed risks and devised a mitigation plan that allowed us to modify our architecture. The iteration two risk analysis document is available in Annex E.3.

This risk assessment phase confirmed that the inability of exporting large maps from the database and compiling them into the OSRM file format, in a short period of time, was the more immediate risk to be dealt with. To mitigate this risk, we planned two activities:

- Convert an array of geographic coordinates into an array of Nodes;
- Conduct Web Routing experiments using the OSRM Traffic Feature.

5.3.2 Convert an array of geographic coordinates into an array of Nodes

To be able to use sensor traffic data, we would have to convert arrays of geographic coordinates into arrays of OSM Nodes. These arrays were necessary to produce the edges contained in the CSV file required for map compilation.

After obtaining random points from the dataset and querying the PostGIS database, we concluded that these points did not translate directly into nodes contained in the database. We could query the database for the closest node to each point but, since we were receiving thousands of points every ten minutes, this process proved too time and resource consuming. We also had no guaranty that the returned node would be present in the OSRM compressed graph.

As an alternative to the PostGIS database, we used the Nominatim geocoding service and OSM Web mapping service API to convert our sensor coordinates into OSM Nodes. Since making 150 reverse geocoding requests a minute to the Nominatim service would be

too time and resource demanding and the coordinate that constituted each path were close together, we opted to calculate the middle point of each line-string and use its coordinates to request the geocoding service for the nearest Way to this point. With the returned information, we would be able to request the OSM service API for the sequence of Nodes that constituted it.

This simple operation allowed us to reduce the number of queries to the Geocoding service by more than 90%. We proceeded to use this information to create the CSV file necessary to compile the OSRM map file, featuring traffic data.

As a consequence of this strategy, the PostGIS database containing the complete OSM map was no longer be necessary. We only needed to have three tables in our data model:

- **Node:** contained the node's *osm_id* and its geographic coordinates;
- **Edge:** contained a source and a target node as foreign keys. By using the sequence of nodes that constituted each Way element returned by the WMS, we were able to construct these edges;
- **Reading:** contained an edge as a foreign key, a speed reading, and a date. Each Edge could have several readings.

Figure 5.3 shows the database relational model that contained all information necessary to run our prototype.

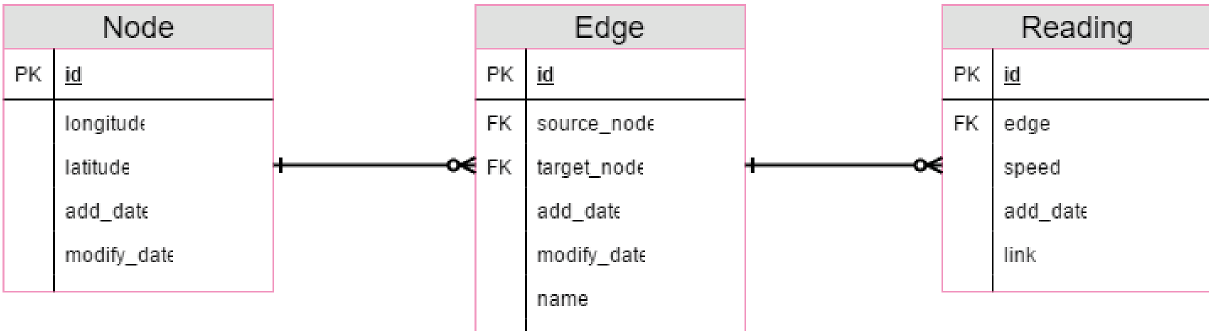


Figure 5.3: Database Relational Model

We built this new data model in the Django project app. Then we adapted our views so they could translate data received from the sensor endpoint into this new model and produce the CSV file necessary for the OSRM map compilation.

We manually compiled the map files and launched the service. Finally, we used the browser-based web client to ensure that the routing engine was working properly using this experimental traffic feature. Since this was the case, we proceeded to the next activity.

5.3.3 Conduct Web Routing experiments using the OSRM Traffic Feature

By editing the CSV file, described in the last section, we reduced the speed of some edges on a given route. After recompiling the map and relaunching the routing service,

it chose a different route. The system seemed to be working properly but this was not conclusive. To reach a more definitive conclusion, we fully automated the process and analyzed the results. We devised two small experiments to determine whether or not sensor data manipulation influenced the distance and time taken to travel from a source to a target location on our map.

To conduct our experiments, we formulated a null and an alternative hypothesis for each of dimensions to be measured:

- **Distance Traveled:**
  - $H0_1$**:** Reducing the max speed value associated with the edges that constitute our map does not increase the distance of the route calculated by our routing engine;
  - $H1_1$**:** Reducing the max speed value associated with the edges that constitute our map increases the distance of the route calculated by our routing engine;
- **Time Taken:**
  - $H0_2$**:** Reducing the max speed value associated with the edges that constitute our map does not increase the time take to travel the route calculated by our routing engine;
  - $H1_2$**:** Reducing the max speed value associated with the edges that constitute our map increases the time taken to travel the route calculated by our routing engine;

To introduce modifications to map data we modified the Cologne traffic data CSV file produced by our system to simulate different traffic conditions of traffic. Each of these files contained 1545 Edges. The speed of each Edge was modified accordingly:

1. **Low traffic:** All Edges had their speed set to 50km/h;
2. **Moderate traffic:** All Edges had their speed set to 25km/h;
3. **Heavy traffic:** All Edges had their speed set to 5km/h;

We compiled map data three times using the OSRM traffic feature and each CSV files we had prepared. Then we deployed three instances of the OSRM routing engine in three separate virtual machines with 2GB of RAM. These instances were identical, only compiled map data varied.

For these experiments, we developed a very simple client script. This script looped the same code an arbitrary number of times. It generated two random geographic coordinates within the city map and queried the services for a path from the source coordinate to the target coordinate. After receiving the result, the script stored it in a CSV file. Each line contained the coordinates, the distance and the time necessary to go from source to target provided by each service. Finally, it scheduled the next cycle. Figure 5.4 represents the architecture of this initial system.

The client script was developed and executed on a forth similar virtual machine. With the necessary system in place, we proceeded to conduct our experiments and retrieved the resulting CSV files.

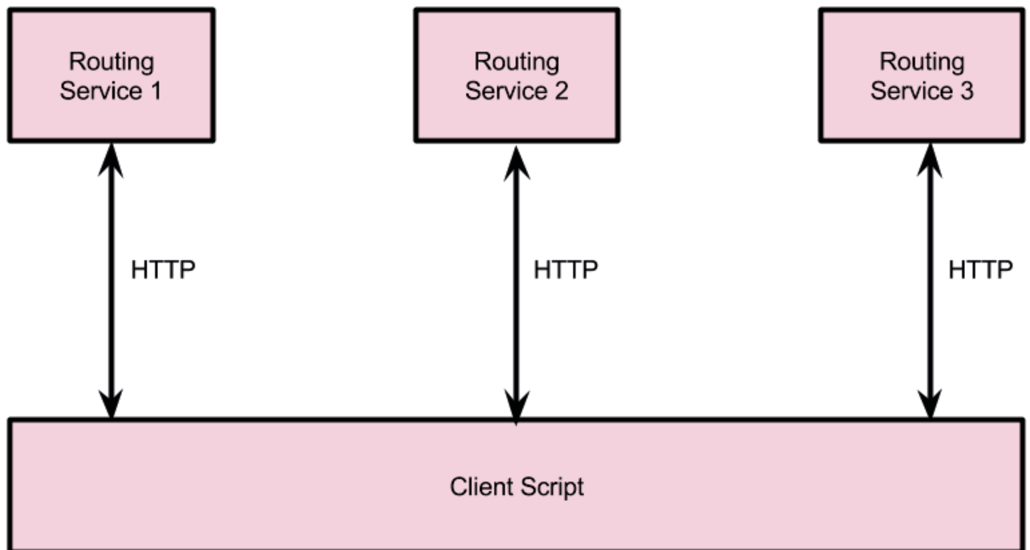


Figure 5.4: Web Routing Experiments System Architecture

5.3.4 Web Routing Experiment 1 - Unmodified Vs Modified Map Export

We deployed an unmodified map Routing Service 1 and a modified map representing heavy traffic in Routing Service 2. Routing Service 3 was not used. After running the client script for **23736 cycles** we obtaining these results:

	Avg Dist	Std Dist	Avg Time	Std Time
Service 1	14269.16 m	7189.82 m	1050.34 s	399.22 s
Service 2	15241.97 m	7743.82 m	1114.05 s	414.9 s

Table 5.1: Experiment 1 Average and Standard Deviation

By applying a paired two-sample t-test, we calculated a two-tailed P-Value of less than 0.001 for both the distance and time variables. The difference between the two samples was very significant. Therefore we should reject both our null hypothesis. In conclusion, reducing the max speed value associated with the edges increased both the distance and the time necessary to go from source to target calculated by our routing engine.

We further analyzed the data obtained to discover what percentage of routes were affected by the differences in the map:

	Serv $1 < 2$	**Serv** $1 = 2$	**Serv** $1 > 2$	**Total**
Distance	40.125%	51.134%	8.741 %	100%
Time	59.387%	40.584%	0.029%	100%
dist && t	40.104%	40.525%	0.021%	80.65%

Table 5.2: Experiment 1 Results distribution

The analysis of table 5.2 showed some interesting results:

- The modified service produced shorter distance paths 8.741% of the time;
- When comparing service one and two results for the same source and target coordinates and taking into account both variables (distance and time), only 80.65% of the time did the comparison signs prove similar;
- In about half of the requests, both servers returned exactly the same distance and time results.

In order justify these unexpected results we further analyzed data:

	$d1 = d2, t1 < t2$	$d1 > d2, t1 < t2$	**Total**
Percentage	10.608%	8.675%	19.283%

Table 5.3: Experiment 1 remaining results distribution

These two particular cases made for almost all the unexpected results, only sixteen elements of the sample remained. Therefore, even though the distance returned by service one could be equal or greater than the distance returned by service two, the time estimation was always better in service one.

Regarding the large percentage of cases in which both services returned the same results, we justified this by the fact that the sensor network only covered a small portion of the roads on the map. Even though these were main roads, it was plausible that in a large percentage of the routes calculated, a sensor covered road was not part of the itinerary.

5.3.5 Experiment 2 - Modified Map Exports: Light vs Moderate vs Heavy Traffic

For this second experiment, we further wanted to analyze how different sensor outputs influenced the distance and necessary time calculated by the services.

We deployed three services:

- **Service 1:** Light Traffic;
- **Service 2:** Moderate Traffic;
- **Service 3:** Heavy Traffic.

We used the same client developed for the first experiment.

After running the client script for **9610 cycles** we obtained these results:

	Avg Dist	**Std Dist**	**Avg Time**	**Std Time**
Service 1	14027.12 m	7044.48 m	1035.52 s	398.78 s
Service 2	14408.97 m	7436.98 m	1059.33 s	405.73 s
Service 3	15096.25 m	7704.61 m	1089.717 s	417.63s

Table 5.4: Experiment 2 Average and Standard Deviation

We applied a paired two-sample t-test two times: service 1 against service 2 and service 2 against service 3, for distance and time variables. We calculated a two-tailed P-Value of less than 0.001. The difference between this three samples was very significant. We reject both null hypotheses.

In conclusion, reducing the maximum speed value associated with the edges that constituted our map, increased both the distance and the necessary time necessary to go from source to target calculated by our routing engines.

5.3.6 Conclusions

During this iteration, we were able to simplify our architecture by using the OSRM traffic feature. After conducting two experiments, even though further verification would be necessary, the results obtained were promising. Whether or not this manipulation could be used to simulate real world traffic conditions remained to be determined.

We adopted the experimental OSRM traffic feature as part of our system. During the next iteration, we integrated it into our architecture and developed a complete system.

5.4 Iteration Three

5.4.1 Planning and Risk Assessment

In the last iteration, we had successfully tested the OSRM experimental traffic feature as an alternative to exporting the map database and compiling the map files for the routing system.

During this iteration, we began by assessing and prioritizing risks. Then we devised

a risk mitigation plan and successfully executed its three activities obtaining our first complete prototype. Finally, we tested our prototype against the GraphHopper traffic data integration demonstration that used the same data.

At this point, we had established that using the OSRM experimental traffic feature was far more desirable than exporting the database and compiling the necessary files for the OSRM service. Although it presented a few challenges, this feature simplified our system as importing the complete OSM into a PosGIS database was no longer necessary.

To be able to use this feature in our final architecture we had to analyze risks and devise a mitigation plan. We will present the most important risks identified during this iteration and the activities that constituted the risk mitigation plan. The iteration three risk analysis document is available in Annex E.4

During this risk assessment phase, we determined that we had no guaranty the Nodes and Edges we were generating by converting the array of coordinates that came from the array were present in the map graph generated by OSRM. If the Edges that contained in the CSV file used to compile the map did not exist in the map graph, the system would not work properly.

Another threat to the project consisted of system validation. We already knew we would not be able to fully test our prototype using validated data in the real world. Nevertheless, we tried to evaluate our system against the GraphHopper traffic data integration demonstration. This service used the same sensor data as our system as input and also calculated the fastest route between a source and a target location taking into account traffic conditions. If we deployed both services and upon being requested a route from the same source to the same target they returned similar replies, it would be an important step towards our system's validation.

During the risk assessment phase, we planned three activities to mitigate risk:

1. Layer the Node Arrays, obtained from the OSM service, on the browser-based web map;
2. Determine if nodes obtained from the OSM service were present on the OSRM map;
3. Evaluate the resulting prototype against GraphHopper traffic data integration demonstration.

5.4.2 Layer the Node Arrays, obtained from the OSM service, on the browser-based web map

After establishing a risk mitigation plan, we started by slightly modifying the front-end server to obtain a GeoJSON representation of Edges stored in the database. Figure 5.5 shows the obtained object layered on a web map using the Geojson tool [Map17]. After comparing these results with figure 5.2, it became clear that using the Nominatim geocoding service and the OSM service to generate the necessary Nodes and Segments was not performing well enough.

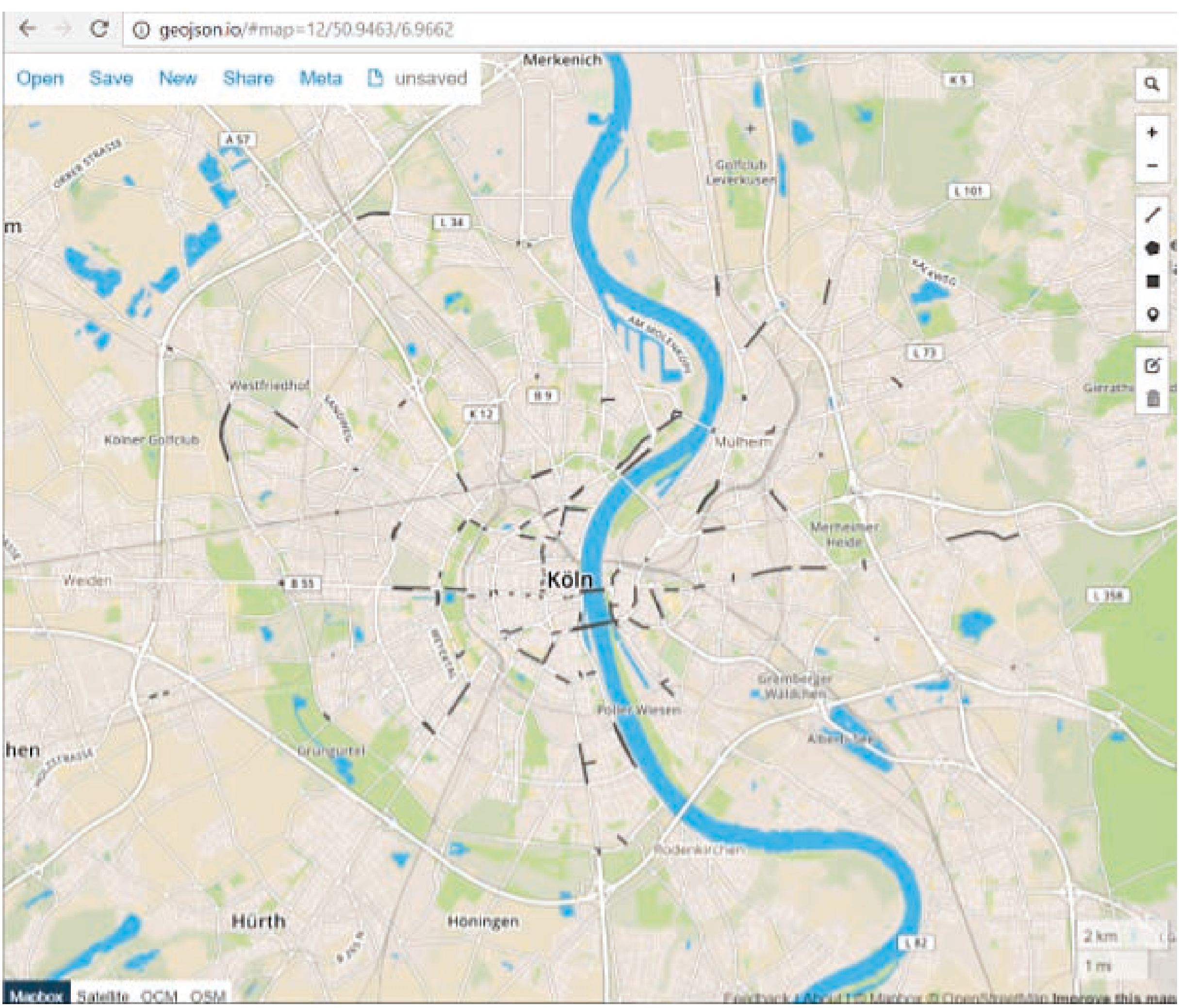


Figure 5.5: Incomplete map layer generated by the Geojson tool

To achieve more coverage we had two options:

- Break the arrays of sensor coordinates into smaller ones, continue calculating the central point of each smaller set and continue to determine the node sequence using the reverse geocoding service and the Web Map Server API. This option produced better results but meant having to make several hundred HTTP requests to these servers every ten minutes;

- Try a completely different approach using a different service like OSRM Match[con17a];

OSRM Match Service: tries match an array of GPS points to the road network in the OSRM map. It removes outliers if they cannot be matched successfully. After some initial testing, this service proved to be much more promising than our initial approach.

Although it took some time to figure out how to interpret the results and match the corresponding coordinates to each node, the results obtained produced more than three times the number of segments of our original approach and a GeoJSON object very similar to our original sensor input. The obtained results were also fully compatible with the existing model represented in figure 5.3.

Using this service, we produced better results with just one request to the OSRM match service for each array of coordinates received from a sensor. This was a huge breakthrough regarding the system scalability. We correctly matched more than 4500 Edges on the OSRM map graph using on average 20 requests to the OSRM match service a minute.

By analyzing the map in figure 5.6, and comparing it to the map generated by the sensor coordinate arrays on figure 5.2, they possess great similarity. By further analyzing discrepancies we concluded that some of them were generated by traffic rules.

Besides being very similar to the original map, this service had the added advantage of using the same compressed map compiled for the OSRM routing service. At this point, we were working with a single map source. By using the OSRM Match service, our application server did not need request information from geocoding or WMSs. Only the browser-based frontend client needed these services.

Working in a highly normalized and compiled map produced by OSRM eliminated redundant information and other discrepancies. This service reduced the risk associated with fulfilling our Geographic Information System necessities with crowd-sourced map data.

5.4.3 Determine if Nodes obtained from the OSM service were present on the OSRM Map

Using the OSRM Match service made the second risk mitigation activity unnecessary. Since we were only working with a single OSRM compressed map instance, all s contained in the Node Array, returned by OSRM match service, were on the map.

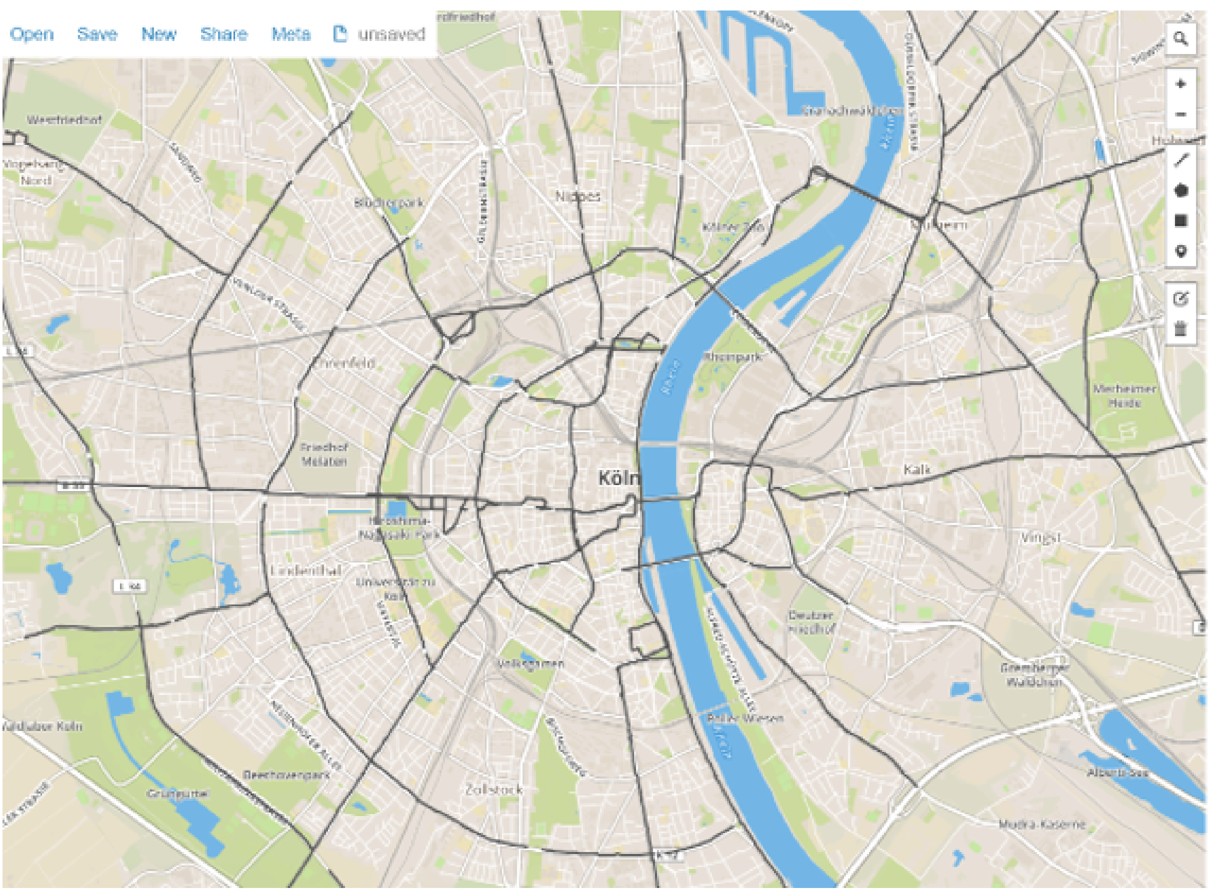


Figure 5.6: Map generated using OSRM match service

5.4.4 Evaluate the resulting prototype against GraphHopper traffic data integration demonstration

Prototype development

On the last activity, we had managed to simplify our system by only using a single map data source for the entire server.

To accomplish this final activity, we developed a complete prototype. This prototype allowed us to evaluate our system by testing it against a completely different service that received the same sensor input. This task was completed with mixed results.

Even though new information became available and we tested new features and services, the high-level architecture of our system did not change. Figure 5.7 shows the high-level architecture for Iteration 3 Prototype.

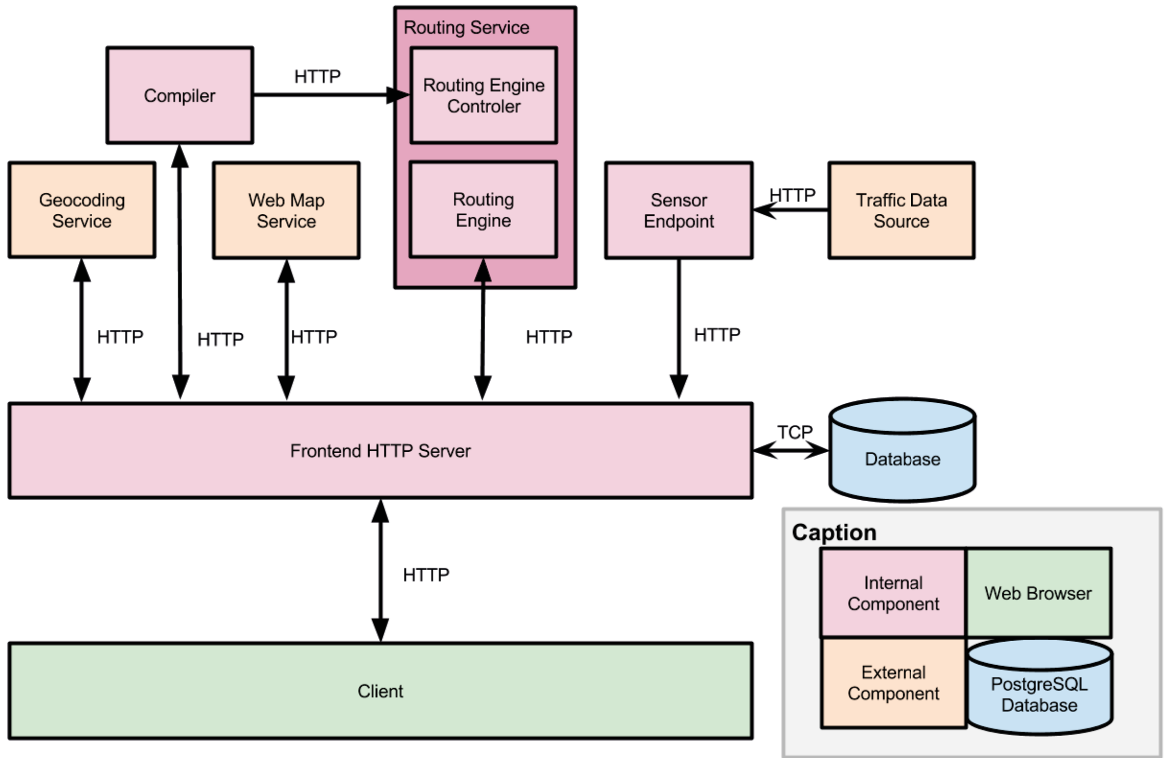


Figure 5.7: Iteration Three System Architecture

Most of the components remained unchanged, we only added two new components to the system:

- **Compiler:** retrieved, from the frontend HTTP server, the traffic information necessary to the OSRM map contraction and stored it in a CSV file. Then it contracted the map and sent the files to the Routing Engine controller using an HTTP POST request;
- **Routing Engine Controller:** acted as a simple HTTP server. It received and processed an HTTP POST request containing map files and stored them in shared

memory. Once these files were available, the OSRM routing engine switched to them.

The proposed architecture was more complicated than necessary, namely the separation of the compiler and the Routing Engine controller in different virtual machines. The downside of this separation was the fact that we had to transfer the compiled files from the compiler to the Routing Service Machine. Nevertheless, since we had modified the limits of shared memory size from the default 64Kilobyte (KB) to 400Megabyte (MB) and allowed users to lock large amounts of shared memory, we considered it was better to isolate the routing service, in a dedicated virtual machine, away from the rest of the system to study its behavior.

Since we were developing a MVP, we did not take into account operational requirements. We were only developing this system to evaluate its feasibility. The compiler was a Python script and Routing Engine Controller was developed using the flask web micro framework [Ron17]. They both executed bash commands from a local OSRM instance.

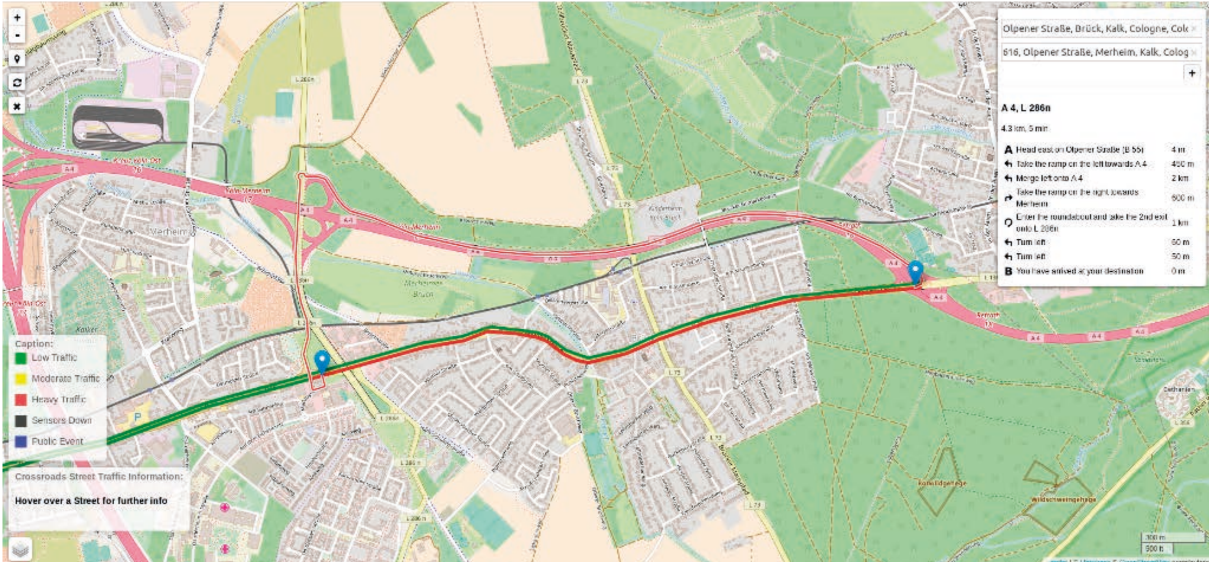


Figure 5.8: Routing results on a very congested route

After constructing our system, we used the browser-based client to visualize how our system was behaving. As expected, traffic was diverted from congested roads, even if that road was the shortest path. Figure 5.8 shows the obtained results.

Although this looked promising, there was still little proof the system was working properly. We proceeded to test it against GraphHopper traffic data integration demonstration that received the same sensor data but took a different approach using this data to influence its routing engine.

The Prototype Vs GraphHoper traffic data integration demo experiment

To better evaluate our prototype, we planned and conducted a small experiment. We slightly modified the client script from iteration one experiences in order to request paths to both systems. Both systems were deployed using the same OSM map and traffic data source. It was our reasoning that, if they received from the client a similar request,

they should produce similar results. Figure 5.9 represents the general architecture of our experiment.

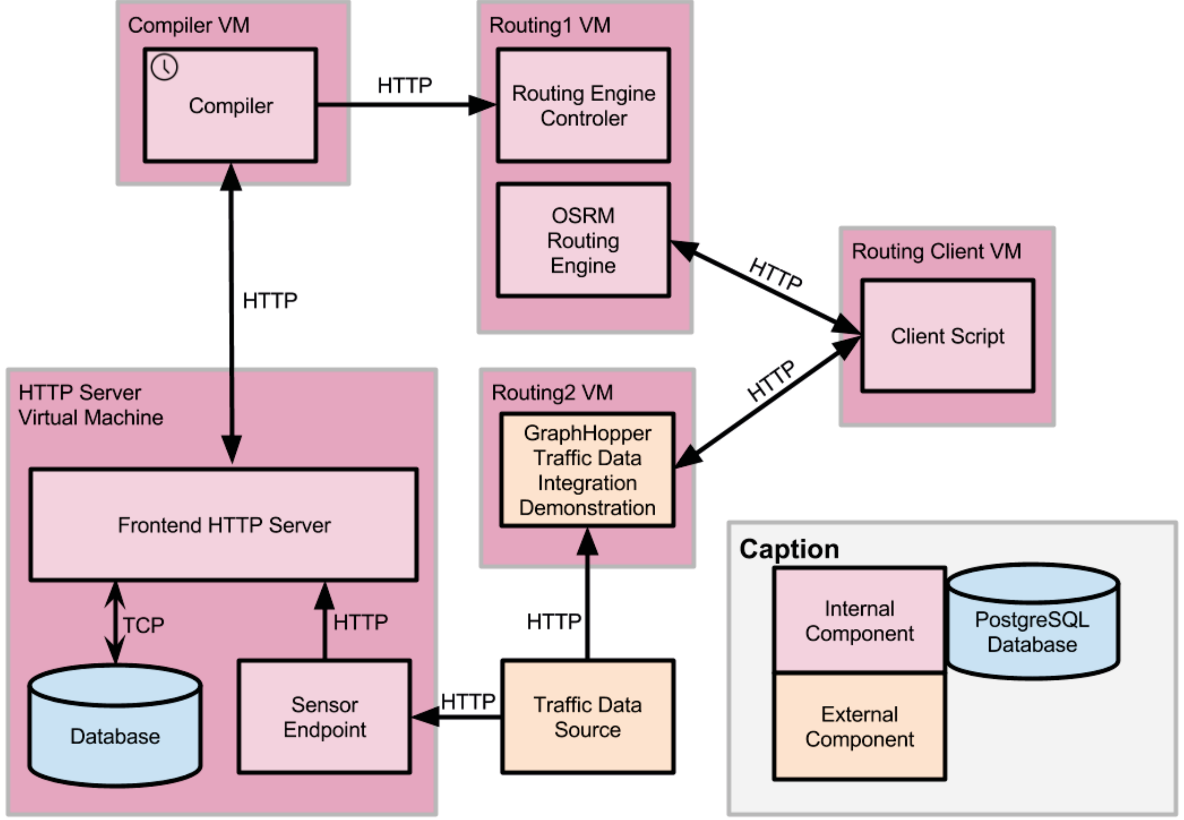


Figure 5.9: Prototype Vs GraphHopper Traffic Data Integration Demo Experience

To conduct our experiments we formulated a null and an alternative hypothesis for each dimension to be measured:

- **Distance Traveled:**
  - $H0_1$**:** Making the similar requests to both routing services, do not produce different replies in terms of distance traveled;
  - $H1_1$**:** Making the similar requests to both routing services, produces different replies in terms of distance traveled;
- **Time Taken:**
  - $H0_2$**:** Making the similar requests to both routing services, do not produce different replies in terms of predicted time to go from source to target location;
  - $H1_2$**:** Making the similar requests to both routing services, produces different replies in terms of predicted time to go from source to target location;

After deploying both systems, the client script executed 800 requests and stored the replies in a file. Then we proceeded to analyze the result. Table 5.5 represents the results obtained.

	Avg Dist	**Std Dist**	**Avg Time**	**Std Time**
Routing1	14064.4m	7140.85 m	874.99 s	339.067 s
Routing2	14109.23 m	7219.06 m	891.33 s	359.706 s

Table 5.5: Prototype Vs GraphHopper Results

We applied a paired two-sample t-test to determine whether or not the difference between these two services was statistically significant.

For the distance dimension, we obtained a t-value of 0.6552 and a P-value of 0.5125. By conventional criteria, the difference between these two samples was not statistically significant. We were able to accept the null hypothesis $H0_1$.

For the time dimension, we obtained a t-value of t = 4.6614 and a P-value of 0.0001. By conventional criteria, the difference between these two samples was extremely significant, We rejected the null hypothesis $H0_2$ and accepted the alternative hypothesis $H1_2$.

In terms of measured distance, both services returned similar results. Regarding time estimation, that did not happen.

After looking for explanations, we concluded that the main culprit should be the vehicle profiles each routing engine used. Since we did not see much point in trying to modify the demonstration's vehicle profiles we decided to consider this activity over.

5.4.5 Conclusions

During this iteration, we developed and evaluated an initial complete prototype. We were able to solve most of the problems related to map data by fulfilling all our geographic information system needs using the OSRM services. By these services, we correctly translated coordinate arrays received from the sensors into Nodes and Edges present in the highly normalized and compressed map instance. This design change also allowed us to do coordinate translation using fewer HTTP requests. This change greatly benefited the projects scalability requirement at the cost of modularity.

Regarding system validation, our experiments had mixed results. We decided not to continue this activity.

Although the produced prototype satisfied all the must have user stories it was far from what was necessary to meet operational requirements. In the next iteration, taking into account lessons learned so far, the entire architecture was revised and a more robust system, that could be accepted by the stakeholders, was developed and tested.

5.5 Iteration Four

5.5.1 Planning and Risk Assessment

In the last iteration, we were able to build a fully functional prototype. Nevertheless, our system was far from respecting all the operational requirements needed to surpass our threshold of success. We would have to rethink our architecture and the relationship between its components. The prototype we had devised in the last iteration was a tightly coupled synchronous system. These characteristics were not desirable in the final prototype. We had to start planning for a system that was modular, portable and extendable.

During this iteration, after assessing risks, we began by revising user stories and the initial architecture. Then we developed the new prototype. Finally, we evaluated our system through several phases of testing. At the end of this iteration, we managed to get the resulting artifacts accepted by the stakeholders.

We assessed risks and reevaluated our project. Using the OSRM match service had allowed using a single highly normalized map source for our project. This breakthrough had taken care of the more urgent risks regarding map data. We decided to leave the system's validation outside the process scope and concentrate on developing a system that met all functional and operational requirements. The iteration four risk analysis document is available in appendix E.5

5.5.2 Engineering and Construction

During the last three iterations, the project evolved significantly. Many of the original assumptions that led to the initial architecture no longer applied. We revised the requirements and architecture to adapt the project to the new information available.

We used YADPT as our project template. Having all these containers already featured and configured using Docker-Compose considerably shortened our development. To build a prototype that could cope with our operational requirements, we added more components:

- **Routing engine:** contained our OSRM services, its controller and the compiler needed to compress the map;
- **Asynchronous task queue worker:** to better scale our system, we had to develop a system of distributed asynchronous tasks based on queues. Since we already had Redis to function as the message broker, we needed to workers to execute tasks. We chose Celery as our asynchronous task job queue;
- **Periodic task scheduler:** we needed a task scheduler to periodically request new data from the sensor data source, schedule the map compilation and backup database information. Since we had already integrated Celery in our system, we used the Celery Beat scheduler;
- **System monitor:** to monitor our queues, tasks, and workers, we needed a system monitor. Since we were already using Celery, we used Celery Cam .

The Web application inside the web container was an empty Django project. We had to develop all necessary apps. Each of these apps contained the models, views, and templates necessary to operate as well as any celery tasks associated with the app.

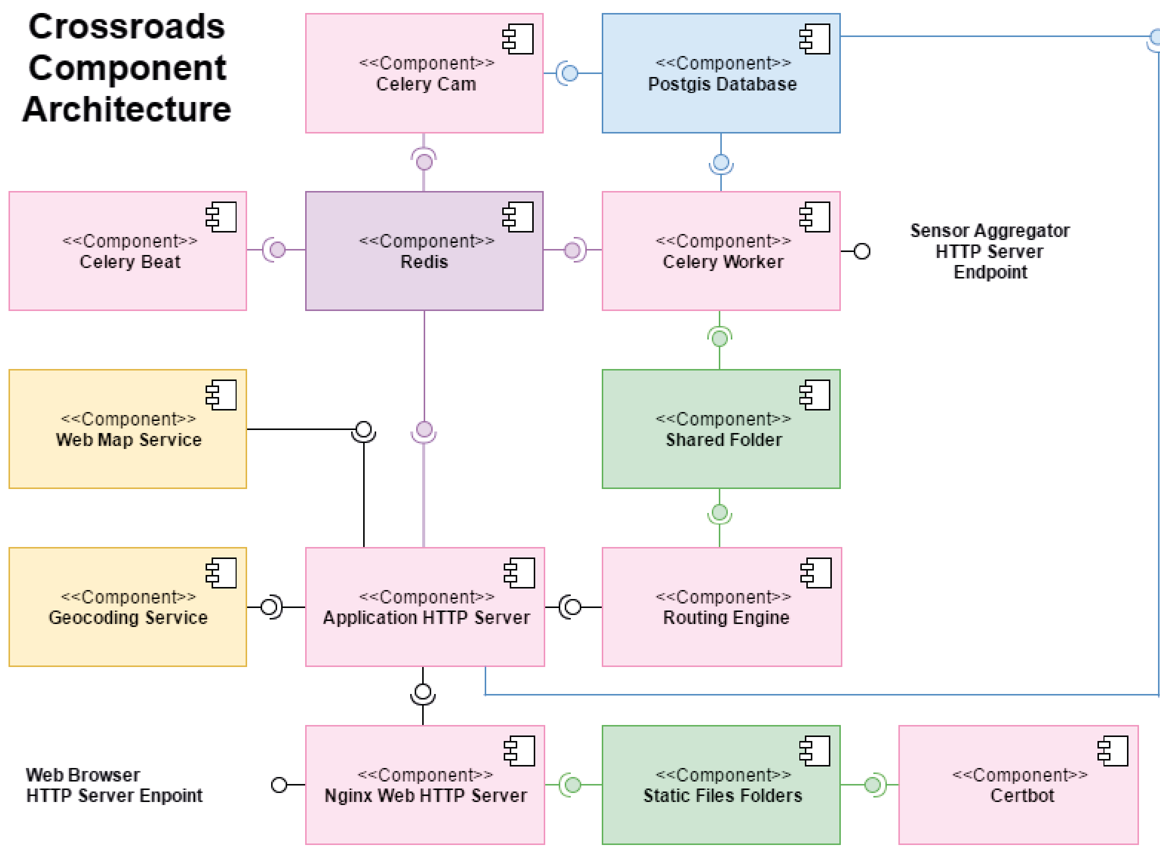
Except for our routing engine, all these components shared the Web Application *Dockerfile*. We only appended the necessary configurations in the docker-compose's YAML files.

Figure 5.10: Iteration 4 Architecture Top View

Figures 5.10 and 5.11 represent the system's architecture. We will now describe each of its components.

Web Application:

When we built YADPT, the web application container featured the necessary settings and requirements to integrate the PostGIS database and Redis. We added all necessary configurations and libraries to integrate Celery into the Django project. With these configurations in place, all we had to do was to develop the necessary tasks and correctly configure Celery Beat and Cam.



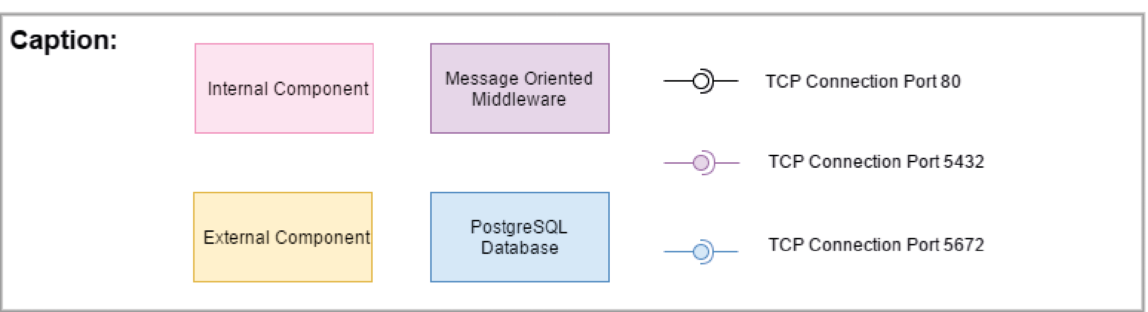


Figure 5.11: Iteration 4 Architecture Component View

The key features of the web application are listed bellow:

- **Separation of the application into two apps:** Since the web application had two very different roles, we split it into two different Django Apps:
  - **Core App:** contained the data model, business logic, and functions related to retrieving and storing sensor data and the Celery tasks;
  - **Frontend App**: featured the views, templates and static contents of the web application as well as the HTTP interface necessary to serve client's requests;
- **Web Browser Client:** The Leaflet based web map client was developed during the first iteration. We made few modifications to integrate it into this final prototype;
- **Administration Console:** Rather than constructing it from scratch, modified the Django Admin web-browser based application. Using this tool, We were able to assess database, tasks, queue, workers information and user management. We used Celery Cam to retrieve this data from the other components and make it available to the console;
- **Celery Tasks:** The Celery workers we deployed with our system did nothing by themselves. To work properly, we needed to develop tasks for them to execute.

Tasks

The most important components of our web application, that would allow our system to operate correctly and meet operational requirements, were the developed tasks. They were thoroughly tested using unit testing. They were able to deal with several exceptional scenarios. For each of these scenarios, we developed a possible solution to prevent tasks from halting or cause inconsistent data insertion in the database.

For this system we developed several tasks:

- **Sensor Endpoint:** requested sensor data to the Cologne traffic data source. For each paths object, it created a Treat Path Group Task to be executed later;
- **Treat Paths:** retrieved all the necessary parameters and paths from the received object. Then it split the paths and created a Get Reading task to be executed later;
- **Get Reading:** split the received path into smaller chunks and queried the OSRM match service. For each reply, it created a Process Reply task to be later execute;
- **Process Reply:** would extract the necessary *osm_id* Node and its corresponding geographic coordinates into an array that could maintain the Node's order. Using this array it would get or create the necessary Nodes from the database. It would then proceed to get or create the necessary edges from the database and, finally, insert a new reading associated with each of the edges;
- **Compose Cache Objects:** extracted the *osm_id* and its corresponding geographic coordinates into an array. Using this array, it got or created all database objects and created a new reading.

The BPMN diagrams that demonstrate how our tasks function are present in Annex F.2.

Both Sensor Endpoint and the Compose Cache Objects were scheduled tasks. The first should run every ten minutes to get the Cologne sensor updates. We scheduled the second because we had no way of determining whether or not all process reply tasks had already been processed.

Routing Engine

This container was developed by creating a *Dockerfile* that imported the original OSRM *Dockerfile* from Docker Hub and adding the necessary libraries to run our Python compiler and controller scripts. Then we developed the necessary script by combining the compiler and the controller components from the last iteration. This script initialized after the OSRM services were available. The service is better described by figure F.3.

Docker-Compose YAML File

Though the YAML files provided by the YADPT already featured part of the necessary containers and established their relations, a lot of modifications would be needed. We started by adding the three Celery containers to both the development and production files. Since Celery was fully integrated with Django, we used the same Dockerfile to build these containers and initialized them differently.

For the Routing engine, we used a separate Dockerfile. We used the docker-compose files to properly configure and synchronize the behavior of the components that were part of this container and make sure the necessary resources were available when it started.

Docker-compose allowed us to share resources such as directories or files between containers. This feature allowed us to share the necessary map files between containers and greatly simplified the system.

Another major modification to the original YAML files was the implementation of a starting sequence for the containers. There was not much point in starting the Celery Containers if the Message Broker was not yet available. Likewise, the Web application needed the database, cache and routing services to be available. For this feature to work properly, we created health checks that, through polling, enabled Docker Compose to assess whether the necessary resources were yet available before launching the remaining containers.

Deployment in production environment

After properly configuring the Docker Compose's YAML production file, we required a properly configured machine and a domain name that could be properly validated to deploy our system in a production environment.

Once these resources became available we started by building our system using Docker-compose. With the containers properly built, we downloaded the necessary OSM map export, extracted, and contracted the OSRM map using the routing engine container. Finally, we deployed the application server for the first time using Gunicorn

WSGI and determined that it was working properly.

With the application server properly serving requests, we configured the NGINX Web Server and generated the necessary certificates to be able to use HTTPS protocol communication. Since NGINX was also used to serve static resources as a reverse proxy, we had to map this resources so it could locate them. After properly configuring NGINX and generating the necessary Let's Encrypt certificates we deployed the entire system without any further issues.

With the system properly deployed, we proceeded to properly evaluate if it respected all functional and operational requirements and could be accepted by the stakeholders. Since we put so much emphasis on tests, we decided to dedicate them a whole section of this report. All conducted tests are described in section 5.6.

5.5.3 Conclusions

During iteration four of the development phase, we updated our architecture, developed and tested our final prototype. The resulting prototype was functionally very similar to the previous one but was far more robust and could satisfy all elicited operational requirements.

Since we put so much emphasis in testing during this iteration, next section will describe all tests that were conducted to verify if our system was working correctly and complied with all functional and operational requirements.

5.6 Testing

5.6.1 Introduction

During the final iteration of our development process, we put great emphasis in testing our prototype. Several types of tests were performed:

- **Unit Testing:** Using the Django testing features, we validated whether our code worked as expected in a multitude of scenarios;
- **Integration Testing:** Using the Django testing features, we studied how the components that composed our system integrated with each other in several different scenarios.
- **Deployment Testing:** After automatically deploying the system using Docker-compose, we assessed whether or not it was properly serving client's requests and how the client performed on different web browsers and resolution;
- **Usability Testing:** We tested our browser-based Web Map client by asking five people not related to the project to perform a series of tasks;
- **Acceptance Testing:** We determined whether or not our system met functional and operational requirements to get accepted by the stakeholders.

5.6.2 Unit Testing

Our prototype was a complex system containing many components from various sources. Nevertheless, most of these components came ”off the shelf” and behaved like black boxes. We only developed the Django application server, the tasks for the distributed task queue and the routing service controller. Therefore, this were the components that would have to be unit tested. All these components were developed using Django, except for the routing engine controller. This web framework allowed us to use its testing features to automate unit testing.

The most difficult part of developing unit tests was determining what we should test. To this end, we followed a very simple set of rules:

- If the code in question is a built-in Python3 function or library, we do not test it;
- If the code in question is a built-in Django class or library, we do not test it;
- If a class contains custom Methods, we must test it;
- If a Model is customized, we must test it;
- If a View is customized, we must test it;
- If the function is a task, it must test it;
- If the code in question is a support class or function developed by us, we must test it.

Following these rules, we began by testing the tasks we had developed. Since these functions called other tasks and made requests to external services, we had to create strategies to test them properly.

Simulating all these possible scenarios without modifying our code required us to be able to mock requests, objects, and functions and patch them over the existing code. To achieve such goal, we used Python’s unittest.mock library [Fou17h]. Using this library, we were able to assert whether or not mocked objects were called or created and control return values completely isolating all data that went in and out of each unit to be tested.

```
(sandbox) mgreis@mgreis-VirtualBox:~/prot3$ docker-compose -f dev.yml run --rm prot3_web bash
Starting prot3_prot3_redis_1
Starting prot3_prot3_postgis_1
Starting prot3_prot3_osrm_routed_1
Starting prot3_prot3_celerycam_1
Starting prot3_prot3_celerybeat_1
Starting prot3_prot3_celery_1
root@09896687197d:/code# python3 manage.py test
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
................................................................................................
----------------------------------------------------------------------
Ran 96 tests in 1.320s

OK
Destroying test database for alias 'default'...
root@09896687197d:/code#
```

Figure 5.12: Unit Testing Results

To unit test our application, 96 tests were developed and executed successfully. We tested each task, function, and view from our web application using a multitude of simulated inputs and mocked external services. Figure 5.12 shows unit testing results.

5.6.3 Integration Testing

After testing each component individually, we proceeded to test how each component would integrate e the others. Starting with the Sensor Endpoint task, we integrated tasks, one at a time, remounting the system. Then we tested the complete system in a multitude of scenarios by mocking the external services. Finally, we removed the mocked services and tested the complete system.

```
root@09896687197d:/code# python3 manage.py test
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
.................
----------------------------------------------------------------------
Ran 17 tests in 0.491s

OK
Destroying test database for alias 'default'...
root@09896687197d:/code#
```

Figure 5.13: Integration Testing Results

Figure 5.13 shows the integration tests results. To further test component integration, we had to deploy the complete system on a dedicated machine. The system proved far too resource intensive to be deployed and properly tested on a virtual machine deployed in a laptop computer.

5.6.4 Deployment Testing

With the system properly deployed we tested whether or not it was working properly. We started by assessing if the Browser-based Client was working properly on the three most common Web Browsers (Google Chrome, Firefox and Microsoft Edge) at different screen resolutions. Then we proceeded to test if the system was serving the remaining HTTP requests and if routing requests with badly formed parameters interfered with the system's normal functioning.

Since our system was not meant for production and we were deploying it in a paid server we did not test how the server would handle heavy loads.

5.6.5 Usability Testing

After determining if the browser-based client was operating properly, we devised a series of tasks to properly test the this client. Five individuals, not involved in the project, were asked to perform them. We measured time necessary for individuals to perform each task. Finally, we took note of any suggestions.

User Task List

For our usability test we created 12 scenarios:

1. Activate the traffic layer;
2. Deactivate the traffic layer;
3. Change the map aspect;
4. Find Kölner Dom, Koeln, NRW, Deutchland;
5. Assess traffic in near Kölner Dom;
6. Find the place on the map with 6.9583° E, 50.9413° N coordinates;
7. Go to directions mode;
8. Find directions from the Lanxess Arena to the Kölner Dom;
9. Find directions from 6.9830° E, 50.9383° N to 6.9583° E, 50.9413° N;
10. Retrace the route;
11. Remove the route;
12. Return to Map Browsing mode.

Task Results

Table 5.6: Task Results

Tasks	User 1	User 2	User 3	User 4	User 5	Avg
1	34s	22s	24s	55s H	29s	33s
2	7s	5s	6s	6s	8s	6s
3	23s	13s	24s	31s	19s	22s
4	65s	40s	52s	67s	56s	56s
5	31s	27s	45s	55s H	37s	39s
6	45s	34s	55s	54s	52s	48s
7	12s	16s	18s	21s	13s	16s
8	66s	49s	76sH	80s H	55s	65s
9	55s	49s	45s	65s	51s	53s
10	17s	29s	25s	28s	22s	24s
11	19s	32s	24s	34s	24s	27s
12	12s	15s	23s	25s	13s	18s
Total	383s	331s	417s	521s	379s	407s

Table 5.6 presents the results obtained for iteration 4 tests. Users were able to accomplish all tasks with minimal intervention from the test supervisor. Our client proved adequate for our prototype. We proceeded to evaluate if our system met functional and operational requirements to be accepted by the stakeholders.

5.6.6 Acceptance Testing

At this point, our prototype had been properly tested and deployed in a production environment. It was time to assess if it could be accepted by the stakeholders. For the project to be a success, it was necessary for our system to meet all functional and operational requirements and restrictions.

Functional Requirements

We defined the project's functional requirements as user stories. During iteration one, when we developed the browser-based client, we made a great effort to include all must have user stories into our client. During the usability test activity, we developed scenarios that took into account these user stories. Since users successfully executed each task, functional requirements were met.

Constraints

All software and external services were completely free of any charge. We used the OSM WMS and Nominatim geocoding service from the OSM Foundation demonstration servers. In the future, to continue system's development, we would have to either buy these services from a third party or deploy and an instance of these services. Nevertheless, we met the restriction.

Non Functional Requirements

During the initial requirement elicitation, eight quality requirements were elicited:

- **Extendability:** This operational requirement had been met by our architecture. The service could be easily extended by adding new Docker service and resource containers, new tasks to the Celery Distributed Task Queue, new Apps to the Django Web Application, new views and models to the Frontend and Core Apps;

- **Interoperability:** The system was easy to integrate with other systems without any modification. The traffic layer data, routing service, traffic GeoJSON object and CSV traffic information were available through HTTPS service. New sensor sources could be added to the system by developing one simple task that acted as its sensor endpoint and translated sensor data into a schema that our system could understand;

- **Modularity:** This requirement was partially met. There were components that could not be traded, namely the OSRM services. This fact was discussed with the stakeholders. We used OSRM with their express approval. The remaining components, like the database and the message broker, could be easily swapped with only minor modifications to the system's configuration files;

- **Portability:** Since Docker was available on all major operating systems, our prototype could be deployed on them. This operational requirement was met;

- **Reusability:** All system components were loosely coupled, communication was achieved using a message broker, and well defined HTTPS interfaces. The Django Web Apps were completely independent of each other. The remaining components were off the shelf. Therefore, the system's components could be easily reused in other present and future services with minimal configuration;

- **Scalability:** The system scaled horizontally. Vertical scalability was off the scope of the project. At this point, the only system's bottleneck was its relational database. We guaranteed Horizontal Scalability at different levels:

  – **Application server level:** The usage of the NGINX Web server allowed for several instances of the application server to be used in parallel to deal with HTTP Dynamic requests;

  – **Celery Worker level:** It was possible to launch more threads and more instances of the Celery Worker to deal with increased task number. These new instances could be easily configured to be deployed on remote machines if necessary;

  – **Routing Engine level:** The service was able to accommodate several local and remote routing services running different vehicle profiles without any major changes to it.

- **Security:** We guaranteed this requirement using the Let's encrypt certificates that allowed the use of HTTPS protocol during Internet communication and, at administration level, by the necessity of authentication to access the console;

- **Usability:** The usability tests conducted had demonstrated that the developed browser-based web map client was adequate for the prototype.

The system met functional and operational requirements only having minor issues in terms of modularity. It was ready to be accepted by the client.

5.7 Conclusions

During this chapter, we described the four iterations taken doing development and tests conducted to evaluate our final prototype. At this point, our project had evolved significantly. The use of OSRM traffic feature and match service had greatly simplified our architecture and allowed our project to meet operational requirements. Without these features, the prototype would struggle with large maps. This limitation could deem an eventual production system unfeasible.

The option to use YADPT as our project template proved adequate. It allowed us to develop a complex system in a period of time and use components of the shelf. The Django web framework, featured in YADPT, also proved ideal for this project. It integrated with the Celery distributed task queue, Redis, and the PostGIS database. Using Django testing feature, We tested our application server without needing any third party tools.

We put a great effort into properly testing our final prototype. Besides the testing done to the application server, we assessed if our application was working correctly after

being deployed in a production environment. Then we performed usability tests to the browser-based client by asking users not related to the project to perform several tasks. Finally, we did an acceptance test to determine if the prototype met requirements and restrictions. At this point, the prototype was ready for acceptance by the client.

In the next chapter, we will discuss if our project met the threshold of success, draw conclusions from the project and explore possible developments in the future.

Chapter 6

Results and Conclusions

6.1 Introduction

In the last chapter, we took four iterations which resulted in a final prototype. This system was fully tested and proved robust enough to meet functional and operational requirements. At the end of testing, we deemed the system ready for acceptance by Ubiwhere.

In this chapter, we will present and discuss the projects result, take conclusions and talk about possible future developments for the system. We will also analyze if we met the project's threshold of success and it was considered successful.

The final artifacts were made available in its private Gitlab Repository to be consulted or used. The code contained a readme file with all necessary instructions to be automatically deployed using Docker-compose. This file was rendered by Gitlab making it easier to consult. This report and the remaining documentation were to be published in the Crossroad's Redmine Project folder.

6.2 Results

At the beginning of the project, we set ourselves to study the feasibility of using sensor data to influence web map routing engines. This rather long process took us 10 months to accomplish and resulted in a final prototype that met all functional and operational requirements and was ready to be delivered to Ubiwhere.

6.2.1 Conducted Activities

We will describe the activities conducted during each semester. During the first semester, we split work into several activities:

1. **Idea Validation:** We studied the motivations behind Crossroads, what were its objectives and what metrics and criteria should be used to measure its success. Then we assessed possible data sources, tools and services that could be used to construct such system. Finally, we deployed and tested these tools and services. This

preliminary work phase gave us a better idea of what our system should accomplish and how we should build it;

2. **Planning:** Since our project was expected to last 10 months, we studied development methodologies. These methodologies helped us plan our activities and promote accountability. By establishing a high-level plan for the project, we organized tasks, estimated the necessary time to accomplish them and used the necessary tools to measure our progress and account spent time;

3. **Requirements elicitation:** With the knowledge gathered at the start of the project, we elicited functional, quality requirements and restrictions. Then we refined our quality requirements by constructing scenarios that allowed us to determine which requirements were architecturally significant;

4. **Initial Architecture definition:** We studied architectural styles that could help us guaranty operational requirements and reached an initial high-level architecture for the project. Finally, we studied technologies that could support this architecture.

At the end of the first semester, we had a fairly defined requirements specification, an initial high-level architecture and a list of technologies that could support it. We knew there were risks for the project, regarding traffic and map data sources, that had to be dealt with as soon as possible. Mitigating these risks was the driving force behind all the second semester's activities.

At the beginning of the second semester, we entered the development phase. We had chosen a risk driven spiral development process. We took four iterations in which we elicited risks, planned and developed activities to mitigate them and evaluated the resulting artifacts at the end of each iteration.

We will briefly describe each of this iterations:

1. **Iteration one:** After establishing the lack of a complete traffic data source as the most urgent risk, we developed a browser-based web map client and the necessary service to support it. This small system allowed us to layer traffic data on the web map to better visualize coverage and data quality. At the end of this iteration, we were able to select and adequate traffic data source for our project. Since our client was very simple, we developed all must have user stories right away.

2. **Iteration two:** Having selected a traffic data source, we reevaluated risks and determined that the problems related with the incapacity to export large web maps on the fly the biggest threat to the project. To mitigate this risk, we conducted a series of experiences that allowed to successfully conclude that the OSRM experimental traffic feature was a better alternative. Using this feature, we simplified our system and allowed it to operate using large maps;

3. **Iteration three:** After identifying risks, we successfully used the OSRM Match service to mitigate these risks. We constructed an initial complete system and tested it against the GraphHopper traffic data integration demonstration with mixed results. Although very different from our system, this demonstration took the same sensor data as our system and produced similar results;

4. **Iteration four:** Although it proved the feasibility of a complete system, the prototype developed during the previous iteration did not meet operational requirements. Taking into account lessons learned from this initial prototype, We built and thoroughly tested a more robust prototype that could be accepted by Ubiwhere.

At the end of the second semester, we possessed a fully tested prototype ready to be accepted by Ubiwhere. From the software engineering standpoint, our development process was successful.

6.2.2 The Browser-based Client Application

The resulting browser-based web map client was a very simple application. It featured a slippy map that could be operated in two modes:

- **Map Browsing mode:** Allowed the user to browse locations on the map using an address or point coordinates. Figure 6.1 represents this mode;
- **Directions mode:** featured the OSRM powered router. By selecting a source and a target location, the application returned the fastest route between this two point taking into consideration traffic conditions. It also gave the necessary directions to go from source to target. This mode is featured in figure 6.2.

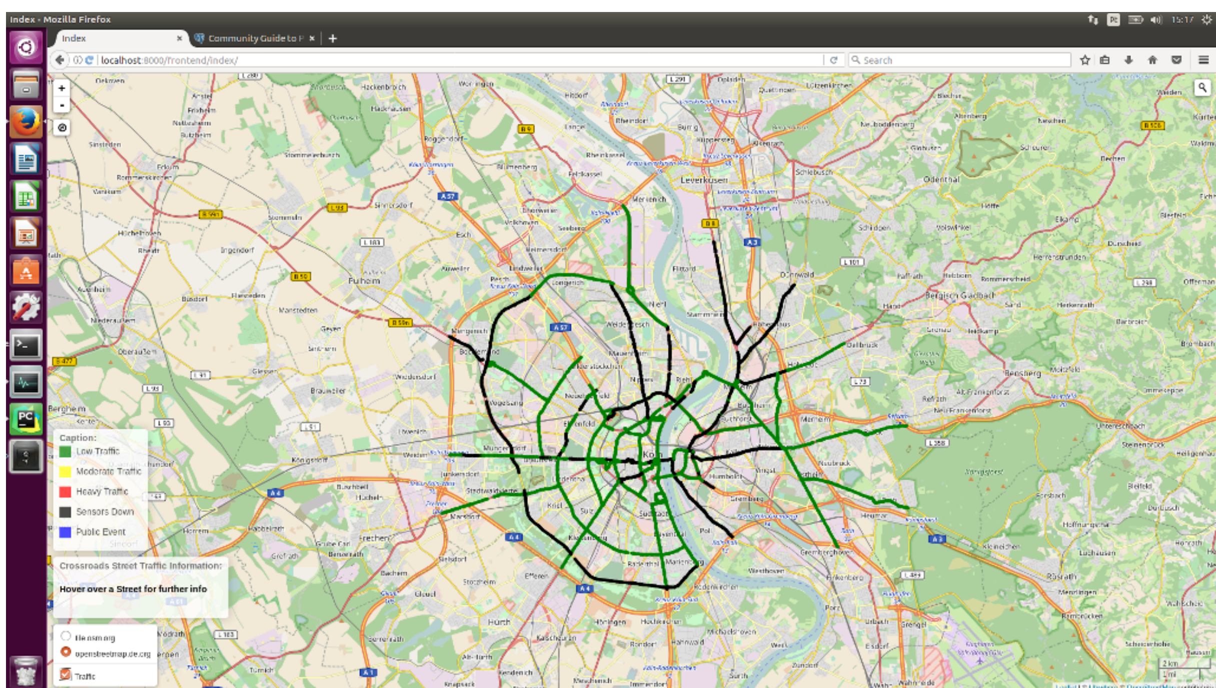


Figure 6.1: Browser-based client in map browsing mode

Besides these modes, the client also featured the necessary form toggle the traffic layer. Although it was simple, this client allowed to properly operate the system and present its capabilities.

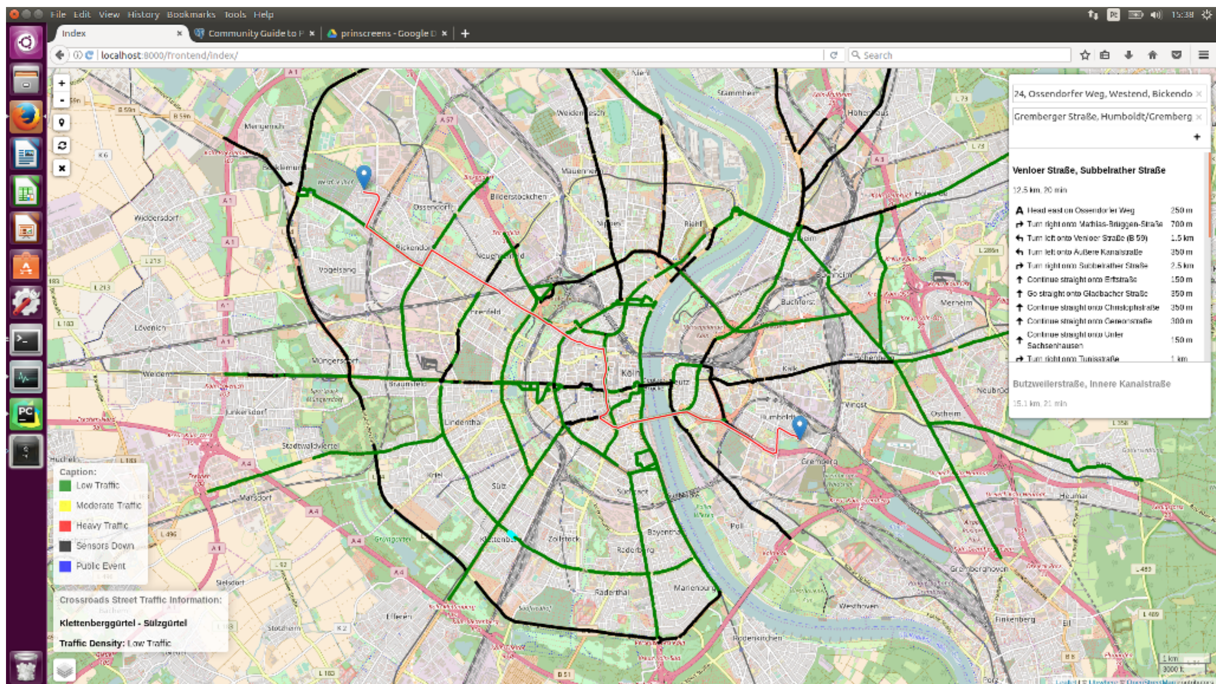


Figure 6.2: Browser-based client in directions mode

6.2.3 Project Evaluation

At the beginning of the project, we had established a threshold of success for our project. We will enumerate these criteria and how they were met:

- The final prototype met all acceptance criteria defined for our Must-Have User-Stories;
- The system respected all defined constraints;
- The system met all the quality requirements defined;
- The Redmine tool reported 1137 hours logged in issues related to the Crossroads project. This value was under the 1176 hours originally allocated for the project;
- The project respected the proposed High-level Plan and Milestones with a less than 2 weeks discrepancy;

Since we met all these metrics and criteria, the project surpassed its threshold of success. At this point, we considered our project to be successful.

6.3 Conclusions

During this project, we analyzed the possibility of integrating traffic sensor data into a routing engine so we could influence how the service calculated the fastest route between a source and a target locations. The information gathered was put to good use in a series of four iterations using an iterative design model which resulted in a series of small prototypes.

From the software engineering standpoint, the process was successful. We were able to elicit requirements, reach and architecture and develop a system that could meet all functional and operational parameters. This system was thoroughly tested and deemed ready for acceptance by Ubiwhere.

From the project's standpoint, the project can also be deemed successful. We met all objectives set for this project and were able to meet all criteria and surpass all metrics defined as the threshold of success for this project.

The technologies and services used to develop our prototypes proved adequate for the project. The OSRM routing service was a valuable asset for our project. The shared memory feature allowed us to update the map being used by the routing engine without having to restart the service. Nevertheless, the experimental OSRM Traffic Feature would also need further testing and validation.

One of the most important breakthroughs during our development phase was the ability drop the OSM map, deployed in a PostGIS database, in favor of the highly normalized and compressed OSRM map format. This breakthrough, made possible by the OSRM match service, simplified our system by allowing it to work with a single map source.

The simplification of our system allowed by the OSRM services, allowed us to store sensor data in a simple, highly normalized format resembling a graph. Since concepts like street addresses or even the street themselves had no use for our system, we stored our data in a format easily exportable into the CSV format used by OSRM during the map contraction step.

If Ubiwhere decides to further develop Crossroads into a product or a product component, lessons learned and artifacts produced during this internship will prove valuable assets to a future development.

From a personal standpoint, This project allowed us to experience a complete development process in a professional environment. We were able to develop and test a complete system, using several new technologies that will prove valuable in the future. This internship was a great opportunity to further hone our skills and will serve us greatly in the future.

Bibliography

[Aga15] Vladimir Agafonkin. *Leaflet - a JavaScript library for interactive maps*. http://leafletjs.com/. (Accessed on 11/07/2016). 2015.

[All17] Andy Allan. *gravitystorm/openstreetmap-carto: A general-purpose OpenStreetMap mapnik style, in CartoCSS*. https://github.com/gravitystorm/openstreetmap-carto. (Accessed on 01/18/2017). Jan. 2017.

[All16] Geographic Information Technology Training Alliance. *Dijkstra Algorithm: Short terms and Pseudocode*. http://www.gitta.info/Accessibiliti/en/html/Dijkstra_learningObject1.html. (Accessed on 10/12/2016). 2016.

[Boe88] Barry W. Boehm. “A spiral model of software development and enhancement”. In: *Computer* 21.5 (1988), pp. 61–72.

[Buc15] Aleks Buczkowski. *Why would you use OpenStreetMap if there is Google Maps? - Geoawesomeness*. http://geoawesomeness.com/why-would-you-use-openstreetmap-if-there-is-google-maps/. (Accessed on 10/03/2016). Oct. 2015.

[Che17] Benoit Chesneau. *Gunicorn - Python WSGI HTTP Server for UNIX*. http://gunicorn.org/. (Accessed on 05/17/2017). 2017.

[Cho+12] H. Chourabi et al. “Understanding Smart Cities: An Integrative Framework”. In: *System Science (HICSS), 2012 45th Hawaii International Conference on*. Jan. 2012, pp. 2289–2297. DOI: 10.1109/HICSS.2012.615.

[Cit17] Citbrain. *Home — Citibrain*. http://www.citibrain.com/en/. (Accessed on 01/23/2017). 2017.

[Cit16a] Citibrain. *Smart Waste Management — Citibrain*. http://www.citibrain.com/en/solutions/smart-waste-management/. (Accessed on 10/07/2016). 2016.

[Cit16b] CityPulse. *CityPulse: Real-Time IoT Stream Processing and Large-scale Data Analytics for Smart City Applications — citypulse*. http://www.ict-citypulse.eu/page/. (Accessed on 01/19/2017). 2016.

[Com12] European Commission. *Road Transport - A change of gear*. 2012. DOI: 10.2832/65952.

[Com16] European Commission. *The European Innovation Partnership on Smart Cities and Communities - European Commission*. http://ec.europa.eu/eip/smartcities/. (Accessed on 10/28/2016). 2016.

[Com17a] European Commission. *Road - European Commission*. http://ec.europa.eu/transport/modes/road_en. (Accessed on 11/03/2016). 2017.

[Com17b] European Commission. *Sustainable transport - European Commission*. http://ec.europa.eu/transport/themes/sustainable_en. (Accessed on 11/03/2016). 2017.

[Com17c] European Commission. *Urban mobility - European Commission*. https://ec.europa.eu/transport/themes/urban/urban_mobility_en. (Accessed on 05/22/2017). May 2017.

[Con16a] Agile Business Consortium. *MoSCoW Prioritisation — Agile Business Consortium*. https://www.agilebusiness.org/content/moscow-prioritisation. (Accessed on 12/12/2016). 2016.

[Con16b] Open Geospatial Consortium. *OGC Standards — OGC*. http://www.opengeospatial.org/docs/is. (Accessed on 11/17/2016). 2016.

[Con16c] Open Geospatial Consortium. *Welcome to the OGC — OGC*. http://www.opengeospatial.org/. (Accessed on 11/17/2016). 2016.

[Con17a] Open Geospatial Consortium. *OpenGIS Web Map Tile Service Implementation Standard — OGC*. http://www.opengeospatial.org/standards/wmts. (Accessed on 01/18/2017). 2017.

[Con17b] Open Geospatial Consortium. *Web Map Service — OGC*. http://www.opengeospatial.org/standards/wms. (Accessed on 01/18/2017). 2017.

[con17a] Project OSRM contributors. *osrm-backend/http.md at master · Project-OSRM/osrm-backend*. https://github.com/Project-OSRM/osrm-backend/blob/master/docs/http.md. (Accessed on 05/22/2017). May 2017.

[con17b] Project OSRM contributors. *Traffic · Project-OSRM/osrm-backend Wiki*. https://github.com/Project-OSRM/osrm-backend/wiki/Traffic. (Accessed on 05/22/2017). May 2017.

[Dij59] Esgar Dijkstra. “A Note on Two Problems in Connexion with Graphs”. In: *Numerische Mathematlk l* (1959), pp. 269–27.

[Dri17] Vincent Driessen. *RQ: Simple job queues for Python*. http://python-rq.org/. (Accessed on 05/16/2017). 2017.

[Eby10] Phillip J. Eby. *PEP 3333 – Python Web Server Gateway Interface v1.0.1 — Python.org*. https://www.python.org/dev/peps/pep-3333/. (Accessed on 05/17/2017). Oct. 2010.

[Eur13] European Innovation Partnership. “European Innovation Partnership on Smart Cities and Communities Strategic Implementation Plan”. In: *European Innovation Partnership on Smart Cities 2013* Strategic Implementation Plan (2013).

[Eur14] European Innovation Partnership on Smart Cities and Communities. “Operation Implementation Plan”. In: (2014), p. 111. URL: http://ec.europa.eu/eip/smartcities/files/operational-implementation-plan-oip-v2%7B%5C_%7Den.pdf.

[Fon17] The jQuery Fondation. *jQuery.ajax() — jQuery API Documentation*. http://api.jquery.com/jquery.ajax/. (Accessed on 05/29/2017). 2017.

[Fou17a] Django Software Foundation. *The Web framework for perfectionists with deadlines — Django*. https://www.djangoproject.com/. (Accessed on 01/17/2017). 2017.

[Fou17b] Electronic Frontier Foundation. *Certbot*. https://certbot.eff.org/. (Accessed on 05/17/2017). 2017.

[Fou15] OpenStreetMap Foundation. *Open Source Routing Machine - OpenStreetMap Wiki*. http://wiki.openstreetmap.org/wiki/Open_Source_Routing_Machine. (Accessed on 11/03/2016). Mar. 2015.

[Fou16a] OpenStreetMap Foundation. *Elements - OpenStreetMap Wiki*. https://wiki.openstreetmap.org/wiki/Elements. (Accessed on 11/28/2016). Aug. 2016.

[Fou16b] OpenStreetMap Foundation. *iD - OpenStreetMap Wiki*. http://wiki.openstreetmap.org/wiki/ID. (Accessed on 01/18/2017). Dec. 2016.

[Fou16c] OpenStreetMap Foundation. *Nominatim - OpenStreetMap Wiki*. http://wiki.openstreetmap.org/wiki/Nominatim. (Accessed on 01/03/2017). Aug. 2016.

[Fou16d] OpenStreetMap Foundation. *OpenStreetMap Foundation Wiki*. http://wiki.osmfoundation.org/wiki/Main_Page. (Accessed on 11/28/2016). 2016.

[Fou16e] OpenStreetMap Foundation. *openstreetmap/openstreetmap-website: Mirror of the Rails application powering http://www.openstreetmap.org*. https://github.com/openstreetmap/openstreetmap-website. (Accessed on 01/18/2017). 2016.

[Fou16f] OpenStreetMap Foundation. *openstreetmap/osmosis: Osmosis is a command line Java application for processing OSM data*. https://github.com/openstreetmap/osmosis. (Accessed on 01/18/2017). Sept. 2016.

[Fou16g] OpenStreetMap Foundation. *PostGIS - OpenStreetMap Wiki*. http://wiki.openstreetmap.org/wiki/PostGIS. (Accessed on 01/03/2017). Aug. 2016.

[Fou16h] OpenStreetMap Foundation. *Tile usage policy - OpenStreetMap Wiki*. http://wiki.openstreetmap.org/wiki/Tile_usage_policy. (Accessed on 12/30/2016). Nov. 2016.

[Fou17c] OpenStreetMap Foundation. *openstreetmap/josm: Git-like mirror of JOSM's Subversion repository*. https://github.com/openstreetmap/josm. (Accessed on 01/18/2017). 2017.

[Fou17d] OpenStreetMap Foundation. *openstreetmap/osm2pgsql: OpenStreetMap data to PostgreSQL converter*. https://github.com/openstreetmap/osm2pgsql. (Accessed on 01/18/2017). Jan. 2017.

[Fc13a] OpenStreetMap Foundation and contributors. *Serving Tiles — switch2osm*. https://switch2osm.org/serving-tiles/. (Accessed on 12/30/2016). 2013.

[Fc13b] OpenStreetMap Foundation and contributors. *switch2osm — Make the switch to OpenStreetMap*. https://switch2osm.org/. (Accessed on 12/30/2016). 2013.

[Fou17e] Python Software Foundation. *Welcome to Python.org*. https://www.python.org/. (Accessed on 01/17/2017). 2017.

[Fou17f] The Apache Software Foundation. *Welcome! - The Apache HTTP Server Project*. https://httpd.apache.org/. (Accessed on 05/17/2017). 2017.

[Fou17g] The Linux Foundation. *About Let's Encrypt - Let's Encrypt - Free SSL/TLS Certificates*. https://letsencrypt.org/about/. (Accessed on 05/18/2017). 2017.

[Fou17h] The Python Software Foundation. *26.5. unittest.mock — mock object library — Python 3.6.1 documentation*. https://docs.python.org/3/library/unittest.mock.html. (Accessed on 06/09/2017). 2017.

[Fou17i] The jQuery Foundation. *jQuery*. http://jquery.com/. (Accessed on 05/29/2017). 2017.

[Git16a] Git. *Git*. https://git-scm.com/. (Accessed on 01/05/2017). 2016.

[Git16b] GitLab. *Code, test, and deploy together with GitLab open source git repo management software — GitLab*. https://about.gitlab.com/. (Accessed on 01/05/2017). 2016.

[Goo16] Google. *Google Terms of Service – Privacy & Terms – Google*. https://www.google.com/intl/en/policies/terms/. (Accessed on 10/03/2016). Oct. 2016.

[Goo17a] Google. *About – Google Maps*. https://www.google.com/maps/about/. (Accessed on 01/18/2017). Jan. 2017.

[Goo17b] Google. *Getting Started — Google Maps Elevation API — Google Developers*. https://developers.google.com/maps/documentation/elevation/start. (Accessed on 05/23/2017). May 2017.

[Goo17c] Google. *Getting Started — Google Maps Geocoding API — Google Developers*. https://developers.google.com/maps/documentation/geocoding/start. (Accessed on 05/23/2017). May 2017.

[Goo17d] Google. *Getting Started — Google Maps Time Zone API — Google Developers*. https://developers.google.com/maps/documentation/timezone/start. (Accessed on 05/23/2017). May 2017.

[Goo17e] Google. *Google Maps Directions API — Google Developers*. https://developers.google.com/maps/documentation/directions/. (Accessed on 05/23/2017). May 2017.

[Goo17f] Google. *Google Places API — Google Developers*. https://developers.google.com/places/. (Accessed on 05/23/2017). May 2017.

[Goo17g] Google. *Introduction to the Google Maps Roads API — Google Maps Roads API — Google Developers*. https://developers.google.com/maps/documentation/roads/intro. (Accessed on 05/23/2017). May 2017.

[Goo17h] Google. *The Google Maps Geolocation API — Google Maps Geolocation API — Google Developers*. https://developers.google.com/maps/documentation/geolocation/intro. (Accessed on 05/23/2017). May 2017.

[Goo17i] Google. *What is Google Map Maker? - Map Maker Help*. https://support.google.com/mapmaker/answer/7278499?hl=en. (Accessed on 01/18/2017). Jan. 2017.

[Gro17a] Internet Security Research Group. *Let's Encrypt - Free SSL/TLS Certificates*. https://letsencrypt.org/. (Accessed on 05/17/2017). 2017.

[Gro17b] The PostgreSQL Global Development Group. *PostgreSQL: The world's most advanced open source database*. `https://www.postgresql.org/`. (Accessed on 06/21/2017). 2017.

[Gut04] Ronald J Gutman. "Reach-Based Routing: A New Approach to Shortest Path Algorithms Optimized for Road Networks." In: *ALENEX/ANALC* 4 (2004), pp. 100–111.

[HW04] Gregor Hohpe and Bobby Woolf. *Enterprise integration patterns: Designing, building, and deploying messaging solutions*. Addison-Wesley Professional, 2004.

[Inc16a] Docker Inc. *Docker - Build, Ship, and Run Any App, Anywhere*. `https://www.docker.com/`. (Accessed on 01/17/2017). 2016.

[Inc16b] Docker Inc. *What is Docker?* `https://www.docker.com/what-docker`. (Accessed on 01/03/2017). 2016.

[Inc17a] Docker Inc. *Docker Compose - Docker*. `https://docs.docker.com/compose/`. (Accessed on 01/18/2017). 2017.

[Inc17b] NGINX Inc. *Welcome to NGINX Wiki! — NGINX*. `https://www.nginx.com/resources/wiki/`. (Accessed on 05/17/2017). 2017.

[Inc17c] NGINX Inc. *Welcome to NGINX Wiki! — NGINX*. `https://www.nginx.com/resources/wiki/`. (Accessed on 05/17/2017). 2017.

[Kar16a] Peter Karussell. *graphhopper/README.md at master · graphhopper/graphhopper*. `https://github.com/graphhopper/graphhopper/blob/master/README.md`. (Accessed on 12/02/2016). Nov. 2016.

[Kar16b] Peter Karussell. *graphhopper-traffic-data-integration/Readme.md at master · karussell/graphhopper-traffic-data-integration*. `https://github.com/karussell/graphhopper-traffic-data-integration/blob/master/Readme.md`. (Accessed on 12/02/2016). Nov. 2016.

[KG15] Omniscale GmbH & Co. KG. *MapProxy — The accelerating web map proxy*. `https://mapproxy.org/`. (Accessed on 01/18/2017). 2015.

[Kha17] Nuno Khan. *django-yadpt-starter 1.3 : Python Package Index*. `https://pypi.python.org/pypi/django-yadpt-starter`. (Accessed on 05/17/2017). 2017.

[KS15] Nikolaos Konstantinou and Dimitrios-Emmanuel Spanos. *Materializing the Web of Linked Data*. Springer, 2015, pp. 111–113.

[La 06] Jeff de La Beaujardiere. "OpenGIS® web map server implementation specification". In: *Open Geospatial Consortium Inc., OGC* (2006), pp. 06–042.

[Lan14] Jean-Philippe Lang. *Overview - Redmine*. `http://www.redmine.org/projects/redmine/wiki`. (Accessed on 01/05/2017). 2014.

[Lei17] Charles Leifer. *coleifer/huey: a little task queue for python*. `https://github.com/coleifer/huey`. (Accessed on 05/16/2017). 2017.

[Lic04] Apache License. "Version 2.0 (http://www. apache. org/licenses/)". In: *You can download the code from* (2004).

[Lie15] Per Liedman. *Leaflet Routing Machine*. `http://www.liedman.net/leaflet-routing-machine/`. (Accessed on 12/02/2016). 2015.

[Lie16] Per Liedman. *perliedman/leaflet-control-geocoder: A simple geocoder form to locate places. Easily extended to multiple data providers.* https://github.com/perliedman/leaflet-control-geocoder. (Accessed on 12/02/2016). 2016.

[Lux13] Dennis Luxen. *High-availability features added to OSRM — Mapbox.* https://www.mapbox.com/blog/osrm-shared-memory/. (Accessed on 11/28/2016). Nov. 2013.

[LV11] Dennis Luxen and Christian Vetter. "Real-time routing with OpenStreetMap data". In: *Proceedings of the 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*. GIS '11. Chicago, Illinois: ACM, 2011, pp. 513–516. ISBN: 978-1-4503-1031-4. DOI: 10.1145/2093973.2094062. URL: http://doi.acm.org/10.1145/2093973.2094062.

[Map17] MapBox. *geojson.io*. http://geojson.io/. (Accessed on 06/22/2017). 2017.

[MPJ10] Joan Maso, Keith Pomakis, and Nuria Julia. "OpenGIS web map tile service implementation standard". In: *Open Geospatial Consortium Inc* (2010), pp. 04–06.

[Mic17] Lda Micro I/O – Serviços de Electrónica. *Micro I/O – Serviços de Electrónica, Lda.* http://www.microio.pt/. (Accessed on 06/21/2017). 2017.

[Ope16] OpenStreetMapFoundation. *Shapefiles - OpenStreetMap Wiki.* http://wiki.openstreetmap.org/wiki/Shapefiles. (Accessed on 01/21/2017). Nov. 2016.

[Ora17] Oracle. *java.com: Java + You.* https://www.java.com/en/. (Accessed on 01/17/2017). 2017.

[Pav16a] Artem Pavlenko. *Mapnik.* https://github.com/mapnik. (Accessed on 12/30/2016). 2016.

[Pav16b] Artem Pavlenko. *Mapnik.org - the core of geospatial visualization & processing.* http://mapnik.org/. (Accessed on 11/28/2016). 2016.

[Pre07] Inc. Present Pivotal Software. *RabbitMQ - Messaging that just works.* https://www.rabbitmq.com/. (Accessed on 01/17/2017). 2007.

[Qui17] Brian Quinion. *twain47/Nominatim: Open Source search based on OpenStreetMap data.* https://github.com/twain47/Nominatim. (Accessed on 01/18/2017). 2017.

[Rie11] Eric Ries. *The lean startup: How today's entrepreneurs use continuous innovation to create radically successful businesses.* Crown Business, 2011.

[Ron17] Armin Ronacher. *Welcome — Flask (A Python Microframework).* http://flask.pocoo.org/. (Accessed on 06/23/2017). 2017.

[Roy70] Winston W Royce. "Managing the development of large software systems". In: *proceedings of IEEE WESCON*. Vol. 26. 8. Los Angeles. 1970, pp. 328–338.

[San17] Salvatore Sanfilippo. *Redis.* https://redis.io/. (Accessed on 05/16/2017). 2017.

[SS05] Dominik Schultes and Peter Sanders. "Highway Hierarchies Hasten Exact Shortest Path Queries". In: *13th European Symposium on Algorithms (ESA).* 2005.

[SC16] European Innovation Partnership on Smart Cities and Communities. *About the partnership - What is it? - European Commission*. http://ec.europa.eu/eip/smartcities/about-partnership/what-is-it/index_en.htm. (Accessed on 10/28/2016). 2016.

[Sc17] Ask Solem and contributors. *Homepage — Celery: Distributed Task Queue*. http://www.celeryproject.org/. (Accessed on 05/16/2017). 2017.

[Sys15] BLIP Systems. *bliptrack_case_studies.pdf*. http://www.eltis.org/sites/eltis/files/case-studies/documents/bliptrack_case_studies.pdf. (Accessed on 01/19/2017). 2015.

[Sys17] BLIP Systems. *BlipTrack Outdoor Sensor : Blip Systems*. http://blipsystems.com/outdoor-sensor/. (Accessed on 01/19/2017). 2017.

[Ubi16a] Ubiwhere. *Ubiwhere — Research and Innovation — Idea to Product — User-centered Solutions*. http://www.ubiwhere.com/en/. (Accessed on 09/29/2016). Sept. 2016.

[Ubi16b] Ubiwhere. *Ubiwhere's Annual Report 2015*. http://www.slideshare.net/Ubiwhere/ubiwhere-2015-annual-report?ref=http://www.ubiwhere.com/en/news/2016/09/01/ubiwhere-annual-report-2015/. (Accessed on 09/29/2016). 2016.

[Ubi17] Ubiwhere. *Up and Running with Django, Docker and Let's Encrypt*. http://www.ubiwhere.com/en/news/2017/03/27/and-running-django-docker-and-lets-encrypt/. (Accessed on 05/17/2017). Mar. 2017.

[UNF16] UNFPA. *Urbanization — UNFPA - United Nations Population Fund*. http://www.unfpa.org/urbanization. (Accessed on 09/29/2016). Sept. 2016.

[WW07] Dorothea Wagner and Thomas Willhalm. "Speed-up techniques for shortest-path computations". In: *Annual Symposium on Theoretical Aspects of Computer Science*. Springer. 2007, pp. 23–36.

[Wav17] Wavecom. *Wavecom The Wireless Experts*. https://www.wavecom.pt/. (Accessed on 06/21/2017). 2017.

[WG06] Renato Werneck and Andrew Goldberg. "Reach for A*: Efficient Point-to-Point Shortest Path Algorithms". In: (2006).

[ZH12] Dennis Zielstra and Hartwig Hochmair. "Using free and proprietary data to compare shortest-path lengths for effective pedestrian routing in street networks". In: *Transportation Research Record: Journal of the Transportation Research Board* 2299 (2012), pp. 41–47.

Appendix A

Shortest Path Problem

Finding the shortest path between two points in a map using a computer system is not a trivial task. We must be able to represent the map in a format that a computer can understand and be able to have an algorithm that can process large maps efficiently in order to find the shortest path.

A.1 Map Representation

Representing a geographical area in order for it to be understood and processed by a computer presents several challenges. Most of the features that humans consider important in a map are of no use to a computer.

All we need for our shortest path calculation are a set of way points and how much effort it takes to get from one way point to the next that is linked to it.

In order to achieve such abstraction directed graphs are a powerful tool.

A directed graph G is an ordered pair:

$$G = (V, A)$$

comprised of a set V of Nodes and a set A of Arrows. These Arrows are themselves oriented pairs of the Nodes that compose the set V.

In this particular implementation all the Arrows A have an associated non negative weight w:

$$w : A \to \mathbb{R}_0^+$$

Finally, In order to find the shortest path in a map we need a source s and a target t:

$$s \in V \land t \in V$$

A.2 Linear Programming Solution

With this directed graph representation, the shortest path from a source s to a target t can be calculated by minimizing the sum of the transversed arrows weight subjected to a constraint condition in which all visited nodes except for s and t must be the destination and source of an arrow contained in the solution.

Given $G = (V, A)$, a source node s, a target node t and an associated weight w_{ij} for each (i, j) arrow:

Consider x a set of variables for whether the edge is part of the solution or not:

$$x_{ij} = \begin{cases} 1 & \text{if } x_{ij} \text{ is part of the solution;} \\ 0 & \text{otherwise.} \end{cases}$$

We must minimize the sum of the weight of the transversed arrows:

$$\min \sum_{i,j \in A} w_{ij} * x_{ij}$$

Subject to a set of restrictions:

$$\text{s.t. } : \forall i \in \ A, \sum_{j \in A} x_{ij} - \sum_{j \in A} x_{ji} = \begin{cases} 1 & \text{if } i = s; \\ -1 & \text{if } i = t; \\ 0 & \text{otherwise.} \end{cases}$$

$$\text{s.t. } : x \geq 0$$

This solution allows for the Shortest Path problem to be solvable in polynomial time. Even though this might be acceptable for graphs with a few hundred vertices, it is not an acceptable solution for larger graphs.

A.3 Dijkstra's Algorithm

Dijkstra's algorithm is used in finding the shortest path between nodes in a given graph. It was created an published by E.W.Dijkstra in 1959 [Dij59].

By considering the graph $G = (V, A)$ the algorithm is supposed to find the shortest path from a given source s to a target t by transposing the arrows A that have a given non negative weight.

There are also variations of the algorithm that allow the shortest path to be calculated from a source node s to any node in the graph.

Pseudocode [All16]: **function Dijkstra(*Graph*, *source*):**

for each *vertex v* in *Graph*: // Initialization

dist[v] **:= infinity** // distance from source to v is set to infinite

previous[v] **:= undefined** // Previous node in path from source

dist[source] := 0 // Distance from source to source

Q **:= the set of all nodes in Graph**

while *Q* is not *empty*: // main loop

u := node in Q with smallest dist[ ]

remove *u* from *Q*

for each neighbor *v* of *u*: // v has not been removed from Q.

alt := dist[u] + dist_between(u, v)

if *alt* <*dist[v]*: // Relax (u,v)

dist[v] := alt

previous[v] := u

return *previous[ ]*

Observation: If only the shortest path from s to t is required, we can stop the search once t is removed from Q.

Discussion: This algorithm returns an array, each of its cells corresponds to a node of the original graph and contains the previous node in the shortest path from the source.

For a given graph this algorithm only has to be run once for each source node. In order to reconstruct the shortest path from the source to any target we only have to use this vector and reconstruct the path backwards. Nevertheless if we only want the shortest path from s to t we can stop the search once t has been removed from Q.

In the worst case scenario, this algorithm has complexity maximized by $O(|V|^2)$. Nevertheless if we do not need the relative distance from each node to the source there are several optimizations that can in practice produce much faster results by reducing the number of visited nodes, although the worst case scenario stays the same.

A.4 Bidirectional Dijkstra Algorithm

If we only want to discover the shortest path from the source to a target node, it might not be necessary to calculate the distance from the source to every node in order to determine the shortest path [WW07].

The bidirectional algorithm shares great similarity to the original algorithm but introduces the concept of frontier which is constituted by the nodes visited so far.

By running the original algorithm twice from source to target and from target to source, one node at a time, we are able to slowly expand this frontier until one scanned node is part of both frontiers. At this point the algorithm stops expanding the frontier.

In order to determine the shortest path from source to target the algorithm sums for each node the distance from the source to it with the distance from it to the target. The shortest path from source to target must path through the node with the lowest sum.

Finally, in order to reconstruct the path, the algorithm uses the previous array like in the traditional case. First of all it reconstructs backwards the path from the source to the chosen node. Then it reconstructs the path from the chosen node to the target. By joining both paths it obtains the shortest path from source to target.

Pseudocode:

First part:

Q_f := **the set of all nodes in Graph for forward search;**

Q_b := **the set of all nodes in Graph for backward search;**

d_t := **distances for forward search;**

d_b := **distances for backward search;**

$previous_f$:= **the set of previous nodes in the shortest path forward search;**

$previous_b$:= **the set of previous nodes in the shortest path backward search;**

While a node has not been removed from both Q_f **and** Q_t**:**

Alternate forward search from s and backward search from t;

Second part:

Find node $\boldsymbol{x}$:

$$x = \min[d_f(x) + d_b(x)], \forall x \in V$$

Third part:

Using $previous_f$ **trace the shortest path from** $\boldsymbol{x}$ **to** $\boldsymbol{s}$**;**

Using $previous_b$ **race the shortest path from** $\boldsymbol{x}$ **to** $\boldsymbol{t}$**;**

Turn the shortest path from $\boldsymbol{x}$ **to** $\boldsymbol{s}$ **backwards;**

Unite the shortest path from $\boldsymbol{s}$ **to** $\boldsymbol{x}$ **with shortest path from** $\boldsymbol{x}$ **to** $\boldsymbol{t}$**;**

Return the shortest path from $\boldsymbol{s}$ **to** $\boldsymbol{t}$**.**

Discussion In the worst case scenario, this algorithm has complexity maximized by $O(|V|^2)$ even though it can run significantly faster than the original Dijkstra algorithm since it visits less nodes.

A.5 Goal Orientated Search (A*)

If our graph G represents a map, we know more or less that an arrow that leads to a node that is nearer to the target node is more likely to be the next step in our path than a node that is located further away. We can therefore say that the node closer to the target has a bigger potential to be part of the solution than the one who is further away.

Taking this empirical knowledge into account it is possible to further optimize Dijkstra's algorithm in order to reach a solution while visiting less nodes, even though the worst case scenario is still maximized by $O(|V|^2)$ [WW07].

This optimization technique modifies the weight of arrows leading from the active node adding a potential (often called heuristic) to its preexisting weight $w(s,t)$. This potential can be calculated by subtracting the potential of the source to the potential of the target:

Given a weighted graph $G = (V, A)$:

$$
\begin{aligned}
& w : A \rightarrow \mathbb{R}_0^+, \\
& \textit{Potential of the source node } p_s : V \longrightarrow \mathbb{R}_0^+, \\
& \textit{Potential of the target node } p_t : V \longrightarrow \mathbb{R}_0^+, \\
& \{s,t\} \in V, \\
& \{(s,t)\} \in A, \\
& \textit{Modified weight } w'(s,t) = -p_t(s) + p_t(t) + w(s,t).
\end{aligned}
$$

If the potential of the target node is lower than the source's the modified weight of the arrow becomes lower than the original one. With the correct potential function, the search can be "pushed" towards the target, greatly reducing the number of visited nodes and consequently its running time.

Special car must be taken into account as the modified weight of a given arrow must always be greater or equal to zero in order to be possible to use Dijkstra's 's algorithm [WW07]:

Given a weighted graph $G = (V, A)$:

$$
\begin{aligned}
& w : A \rightarrow \mathbb{R}_0^+, \\
& \textit{a potential } \ p : V \longrightarrow \mathbb{R} \ \textit{ is called feasible,} \\
& \textit{if } \ w(s,t) - p(s) + p(t) \geq 0, \forall a \in A.
\end{aligned}
$$

There are several strategies to calculate feasible potentials:

Euclidean Distances. Lets assume that the weight of an arrow is somewhat correlated with the Euclidean distance between its end nodes. By calculating the euclidean distance between each node of the graph and the target node we get a series of feasible potentials.

Due to the triangular inequality principle we know that our potentials are always feasible:

$$\textit{if } w(s,t) - p(s) + p(t) \geq 0, \forall a \in A.$$

Since we also know that a node nearer to the target will always have a lower euclidean distance to it than a node farther away, our search will always be pushed toward the target. In order to further speed up the process, the computationally expensive operation of calculating the Euclidean distance, due to the existence of a square root, can be replaced by an approximation.

Landmarks. Lets again assume that the weight of an arrow is somewhat correlated with the Euclidean distance between its end nodes. Since before calculating any path we already have all the information about the graph G, this information can be used in order to pre compute and store the distance between every node $v \in V$ of the graph and a small fixed- sized subset $L \subset V$ of chosen landmarks.

For each landmark $l \in L$ we can define potential as: [WW07]

$$p_t^{(l)}(v) = d(v, l) - d(t, l)$$

This potential p_t^l is always feasible and a lower bound for the distance between v and the target t , due to the triangle inequality:

$$d(v, l) \leq d(v, t) + d(t, v)$$

Since any p_t^l calculated is feasible and a lower bound we can easily define a potential p_t for a certain node $v \in V$ as:

$$p_t(v) = \max\{p_t^{(l)}(v) : \forall l \in L\}$$

Given certain path search, nodes that are situated near or "behind" the target, constitute good landmarks as the shortest path from v to the target node or the landmark probably share a common sub path. Landmarks located in other areas of the graph might attract the search to themselves [WW07].

Therefore, a great deal of care must be taken into account when selecting landmarks only considering landmarks with the highest potential. This also has the advantage of simplifying the calculation and making it faster.

Distances from graph condensation. If the weight of each arrow of a given graph has to take into account several factors that can change, it might become too complicated to weight all arrows of the graph. Therefore we can pre- calculate a lower bound by calculating the minimum possible weight from each node of the graph to the target node by running the complete Dijkstra's algorithm on a condensed graph.

We can then use the obtained array and use it as a feasible potential set for the expanded graph, making the search much faster, as less nodes are visited and there are less weights to be calculated on the spot.

A.6 Hierarchical Methods

This method requires some level of pre- processing in which a given graph $G = (V, A)$ is populated with additional arrwos that represent the shortest path between the two nodes

that constitute this arrows.These additional arrows produce new levels that coarsen the graph.

In order to find the shortest path between two nodes s and t, we can use Dijkstra's algorithm over a very condensed sub graph at a higher hierarchy and a set of upward and downward arrows.

There are two main methods to deploy this approach:

Multi-level Approach. A given graph $G = (V, A)$ is decomposed using separators $S_i \subset V$ called *selected nodes*. A node contained in a certain level is also contained in lower levels

The selection of separator nodes can be made using diverse criteria like the node's degree in a graph though better criteria can be found.

Arrows in a graph become of three different types:

- *Upward Arrows:* going from a node that is not selected at one level to a node selected at that level;
- *Downward Arrows:* from a selected node of a given level to a non selected node;
- *Level Arrows:* an arrow between two selected nodes at one level. The weight of this edge is assigned the length of the shortest path between this two nodes. Alternatively a large number of small sub graphs can be calculated instead of a large multilevel graphs. This smaller graphs can be optimized individually and produce smaller query times at the cost of heavier processing.

Highway Hierarchies. When we travel, we normally use the main roads in order to reach the general area of our destination. Only when we start our trip or arrive at the desired neighborhood, we use use the secondary roads.

Shortest path trees are used to determine a hierarchy. This has the advantage that no additional information like a separator is needed. By modifying Dijkstra's algorithm, we can assure that, if we want to go from node s to node t with a shortest path $s, u_1, u_2, ..., u_{n-1}, u_n, t$; the sub- path $u_1, ..., u_n$ is always returned as the shortest path between u_1 and u_n. These shortest paths are called *canonical* [SS05].

The process described in the last paragraph can be repeated several times resulting in further graph contraction and new levels of hierarchy.

A.7 Node and Edge Labeling

Reach-Based Routing This type of routing prunes the search tree based o a centrality measure called "reach" [Gut04].

Reach:

Given:

A weighted graph $G = (V, A)$,
$w : A \to \mathbb{R}_0^+$,
$P :=$ *shortest path between* s *and* t *where* $s, t \in V$,

The reach of a node v where $v \in P$ is defined as:
$r(v, P) := min\{w(P_{sv}), w(P_{vt})\}$

The reach *r(v)* where $v \in V$ and $v \in P$:
$r(v) := \max\{r(v, P)\}$

While searching for the shortest s-t path p_{st}, a certain node $v \in V$ can be ignored if:

- $w(P_{sv}) > r(v)$;
- $w(P_{vt}) > r(v)$;

The first item is easy to know since $w(P_{sv})$ is already calculated. For the second item a suited heuristic as the ones described in A* should be used.

In order to compute the reach for every node in a graph, we must perform a single-source all target shortest-path for every node [WW07]. This is easily achieved using a modified depth first search on the shortest path trees with the following insight:

Given two shortest paths P_{sx}, P_{sy} with a common node v:

$$\max\{r(v, P_{sx}), r(v, P_{sy})\} = min\{w(P_{s,v}), \max\{w(P_{vx}), w(P_{vy})\}\}.$$

The computation of reach is maximized by $O(n^2 * \log n)$ time and $O(n)$ space for sparse graphs. In situations which such heavy pre- processing is not acceptable, upper bounds for reach can be calculated [Gut04].

Arrow Labels. This approach labels each arrow with a list of nodes to which a shortest path starts with this particular arrow:

Given the graph $G = (V, A)$;

For each arrow $a = (u, v), a \in A$:

The set S of all nodes $t \in V$: The shortest u to t starts with a;

The shortest path problem is them answered by running Dijkstra's algorithm restricted to those arrows to which the target node is in the solution S. Each arrow label shows the algorithm whether or not the target node might be in the target region of the arrow.

Geometric Containers. Storing all labels *S(u,v)* that result from Arrow labeling would take an amount of space maximized by $O(n^2)$ [WW07]. In alternative it is possible to store everything in a super set that can be represented with constant size and grows linearly.

Given a layout $L : V \longrightarrow \mathbb{R}_0^+$, a *Bounding box* that contains $\{L(t)|t \in S(u, v)\}$ is a very efficient geometric container. This bounding boxes can be calculated beforehand by running a single-source all-target shortest-path computation for every node. This operation requires $O(n^2 * \log n)$ time and $O(n)$ space [WW07].

Arc Flags. This approach partitions the node set in p regions with the function $r : V \longrightarrow \{1, ..., p\}$. Then these p regions each be represented by a bit in a vector. For a given arrow $a \in A$, a region is marked in its bit vector if it contains a node $v \in V$ that belongs to the set of a ($v \in S(a)$). With this approach, since we are using regions, we do not need to compute all-pairs shortest paths.

Every shortest path from a node $s \in V$, outside a region $R \in V$, to a node $t \in R$ has to enter the region R at some point. Since s is not a part of the region, there has to be some arrow $a = (u, v)$ such that u is outside the region R and v is inside. Therefore, it is enough for the pre- processing algorithm to calculate the shortest path to node v at the boundary of the region, making the pre- calculation less computationally intensive with the results occupying less space.

A.8 Combining techniques

By using the techniques described we are able to significantly speed up our shortest path search even though the worst case scenario is still maximized by $O(n^2)$ time. Since these techniques operate over different aspects of the graph and search algorithm it is possible to further speed up the search by combining methods.

Bidirectional Search and Goal-Directed Search. Very hard to use, since selecting the right potential for both search directions is very hard. The expanded frontiers might not converge fast enough as the heuristic might lead the frontier expansion almost to the source of the other direction before a node is removed from both queues;

Bidirectional Search with Hierarchical Methods. Requires a symmetric, backward version of the sub graph to be implemented;

Bidirectional search and Reach Based Routing. the reach criteria can be used directly in backward search. Furthermore, since we are also doing a backward search, we have more information and don't have to use a lower bound to complete the missing information;

Bidirectional Search and Edge Labels. In order to use bidirectional search a second set of edge labels must be computed;

Goal-Directed Search and Highway Hierarchies. The original highway algorithm already accomplishes a bidirectional search [SS05]. The algorithm can be further enhanced by using individual potentials for forward and backward search on landmarks.Unfortunately, the highway algorithm cannot abort the search as soon as an s-t path is found. However, another aspect of goal-directed search can be exploited: the pruning [WW07];

Goal-Directed Search and Reach-Based Routing. Goal-directed search can also be applied to the subgraph that is defined by the reach criterion [WW07]. The combination

of these two strategies involves choosing landmarks and computing reaches. These two procedures are independent from each other: since shortcuts do not change distances, landmarks can be generated regardless of what shortcuts are added [WG06].

Appendix B

User Stories

This Annex contains the elicited user stories as well as the necessary acceptance conditions for our project

B.1 Players/stakeholders

1. **End-Users**
    1.1. **End-User:** in order to use the system's web map features authentication is not necessary. this player has access to all functionalities of the front-end application and the administration log in web page;
2. **Operating Entities**
    2.1. **System Super User:** is responsible for deploying and operating the system and manage Operators.
        2.1.1. **Unauthenticated Super User:** only has access to all functionalities of the front-end application and the administration log in web page;
        2.1.2. **Authenticated System Super User:** has access to all all administration functionalities;
    2.2. **System Operator: Is responsible for operating the System;**
        2.2.1. **Unauthenticated System Operator:** only has access to all functionalities of the front-end application and the administration log in web page;
        2.2.2. **Authenticated System Operator:** has access to all administration functionalites that were authorized by the Super User for his use.

B.2 Work Division Structure

1. End-User using the front-end application GUI
    1.1. Map Interface Manipulation
    1.2. GUI Manipulation
        1.2.1. Search Mode
            A. Address Translation
        1.2.2. Directions Mode
            A. Address Translation
2. Unauthenticated System Operator and Super User GUI
    2.1. Administration Log in
3. Authenticated Super User GUI
    3.1. Administration Dashboard
4. Authenticated System Operator GUI
    4.1. Administration Dashboard

B.3 User Stories Structure

- **Description** - As a [user role] I want to [goal] so I can [reason].
- **Acceptance Criteria** - User must be able to [action that describes the functionality]
- **User Story Numbering Scheme** - US-[WBS-Submodule-ID]-#

B.4 User Stories Definition

[1] End-User front-end application GUI

[1a] Map Interface Manipulation

US-1a-1.

As an End-User I want to manipulate the map zoom and visualization window so I can better visualize the desired map elements.

Acceptance Criteria:

1. User must be able Zoom in and out on the map;
2. User must be able to move the map horizontally and vertically;

US-1a-2.

As an End-User I want to be able to visualize traffic density so I can better assess the present traffic situation.

Acceptance Criteria:

1. User toggle a traffic density map layer on and off by pressing a button;
2. User must be able to assess live traffic information by observing the graphic map interface;

US-1a-3.

As an End-User I want to be able to visualize air quality so I can better assess the present the present environmental situation.

Acceptance Criteria:

1. User toggle a air quality map layer on and off by pressing a button;
2. User must be able to assess live air quality information by observing the graphic map interface;

US-1a-4.

As an End-User I want to be able to visualize available parking density so I can better assess the present parking availability situation.

Acceptance Criteria:

1. User toggle a parking availability layer on and off by pressing a button;
2. User must be able to assess live available parking density information by observing the graphic map interface;

[1b]GUI manipulation

US-1b-1.

As an End-User I want be able to swap between two GUI modes (Search and Directions) so I can better manipulate the interface with different behaviors and functionalities.

Acceptance Criteria:

1. User must be able to swap from Search mode to Directions mode;
2. User must be able to swap from Directions mode to Search mode;

[1bi] Search mode

US-1bi-1.

As an End-User I want be able to select a new point on the map each time I press the mouse button.

Acceptance Criteria:

1. User must be able to select a new point on the map each time he presses the mouse button;

[1biA] Address Translation

US-1biA-1.

As an End-User I want to translate any set of valid geographic coordinates (Latitude and Longitude),inserted in a valid form, into a valid point on the map so I can find any geographic location.

Acceptance Criteria:

1. User must be able to locate any point on the map using a set of valid geographic coordinates.

US-1biA-2.

As an End-User I want to translate any set of valid geographic coordinates (Latitude and Longitude),inserted in a valid form, into a valid street address if one exists so I can find any address from a given set of coordinates corresponding to it.

Acceptance Criteria:

1. User must be able to find any valid address using a set of valid geographic coordinates that correspond to it;
2. User must be able to locate any valid address in the map using a set of valid geographic coordinates that correspond to it.

US-1biA-3.

As an End-User I want to translate any valid address,inserted in a valid form, into a valid point on the map so I can find any element on the map.

Acceptance Criteria:

1. User must be able to find any element on the map using a valid address that correspond to it;
2. User must be able to choose from several valid addresses if the address provided is ambiguous.

US-1biA-4.

As an End-User I want to select with the pointing device any point on the map and translate it into a valid set of geographic coordinates and a valid address if one exists so I can retrieve information about any point on the map without knowing its address or geographic coordinates.

Acceptance Criteria:

1. User must be able to find the geographic coordinates of any point on the map by selecting it.
2. User must be able to find the address of any point on the map, if that address exists, by selecting it.

[1bii] Directions mode

US-1bii-1.

As an End-User I want to be able to select a source and a target on the map so I can obtain the fastest path from the source to the target.

Acceptance Criteria:

1. User must be able to select a source when he presses the mouse on the map for the first time;
2. User must be able to select a target when he presses the mouse on the map for a second time;
3. User must receive a route when after a target point is selected;
4. User must be able to drag the source point to a new location and receive a new route after the mouse button is dropped;
5. User must be able to drag the target point to a new location and receive a new route after the mouse button is dropped;

US-1bii-2.

As an End-User I want to be able to insert valid addresses into a source and a target valid form so I can obtain the fastest path from the source to the target.

Acceptance Criteria:

1. User must be able to insert a source address into a valid form;
2. User must be able to insert a target address into a valid form;
3. User must receive a route when after both source and target addresses have been inserted into the form;
4. User must be able insert a new valid address into the the source form and receive a new route;
5. User must be able insert a new valid address into the the target form and receive a new route;

US-1bii-3.

As an End-User I want to be able to insert valid geographic coordinates into a source and a target valid form so I can obtain the fastest path from the source to the target.

Acceptance Criteria:

1. User must be able to insert valid source geographic coordinates into a valid form;
2. User must be able to insert valid source geographic coordinates into a valid form;
3. User must receive a route when after both source and target coordinates have been inserted into the form;
4. User must be able insert new valid geographic coordinates into the the source form and receive a new route;
5. User must be able insert new valid geographic coordinates into the the target form and receive a new route;

US-1bii-4.

As an End-User I want to be able to choose from several mobility profiles so I can choose which mean of transportation I will use to reach my target.

Acceptance Criteria:

1. User must be able to select driving as a mode of transport;

US-1bii-5.

As an End-User I want to be able receive directions from source to target after my route has been plotted so I can better navigate the route.

Acceptance Criteria:

1. User must be able to see the entire route after its been plotted.
2. User must be able to see written directions, organized from source to target after the route has been plotted.

US-1bii-6.

As an End-User I want to be able choose from several routing types so I can better plan my trip.

Acceptance Criteria:

1. User must be able to select fastest path as a routing type if enough information about the area is available.

[1biiA] Address Translation

US-1biiA-1.

As an End-User I want to translate any set of valid geographic coordinates (Latitude and Longitude),inserted in a valid form, into a valid point on the map so I can find any geographic location.

Acceptance Criteria:

1. User must be able to locate any point on the map using a set of valid geographic coordinates.

US-1biiA-2.

As an End-User I want to translate any set of valid geographic coordinates (Latitude and Longitude),inserted in a valid form, into a valid street address if one exists so I can find any address from a given set of coordinates corresponding to it.

Acceptance Criteria:

1. User must be able to find any valid address using a set of valid geographic coordinates that correspond to it;
2. User must be able to locate any valid address in the map using a set of valid geographic coordinates that correspond to it.

US-1biiA-3.

As an End-User I want to translate any valid address,inserted in a valid form, into a valid point on the map so I can find any element on the map.

Acceptance Criteria:

1. User must be able to find any element on the map using a valid address that correspond to it;
2. User must be able to choose from several valid addresses if the address provided is ambiguous.

US-1biiA-4.

As an End-User I want to select with the pointing device any point on the map and translate it into a valid set of geographic coordinates and a valid address if one exists so I can retrieve information about any point on the map without knowing its address or geographic coordinates.

Acceptance Criteria:

1. User must be able to find the geographic coordinates of any point on the map by selecting it.
2. User must be able to find the address of any point on the map, if that address exists, by selecting it.

[2]Unauthenticated System Operator and Super User GUI

[2a] Authenticate an Unauthenticated System Operator

US-2a-1.

As an Unauthenticated System Operator or Super User I want to be able to log in to the system so I can log in and access all system functionalities.

Acceptance Criteria:

1. User must be able to access the Login page;
2. User must be able to fill in a form containing:
   2.1. his email;
   2.2. a password.
3. User must be able to submit a correctly filled form in order to get logged in;
4. User must advance to the application main page to the user if log in was successful;
5. User must receive an error message if log in was unsuccessful.

[3] Authenticated Super User GUI

[3a] Administration Dashboard

US-3a-1.

As an Authenticated Super User I want to be able to manage logs so I can better assess and manage system events.

Acceptance Criteria:

1. User Super User must be able to manage log entries:
   1.1. Add Log Entries;
   1.2. Change Log Entries;
   1.3. Delete log entries.

US-3a-2.

As an Authenticated Super User I want to be able to manage authorizations so I can better assess and manage system operators.

Acceptance Criteria:

1. Super User must be able to manage user groups:
   1.1. Add User Groups;
   1.2. Change User Groups;
   1.3. Delete User Groups.
2. Super User must be able to manage permissions:
   2.1. Add permission;
   2.2. Change permission;
   2.3. Delete permission.
3. Super User must be able to manage System Operators:
   3.1. Add System Operator;
   3.2. Activate System Operator;
   3.3. Deactivate System Operator;
   3.4. Allow System Operator to log to the system;
   3.5. Disallow System Operator to log to the system;
   3.6. Change System Operator;
   3.7. Delete System Operator.
   3.8. Manage System Operator and groups:
      3.8.1. Add System Operator to group;
      3.8.2. Remove System Operator from group.
   3.9. Super User must be able to manage Super Users:
      3.9.1. Promote System Operator to Super User;
      3.9.2. Demote Super User to System Operator.

US-3a-3.

Asan Authenticated Super User I want to be able to manage database tables so I can better assess and manage stored information.

Acceptance Criteria:

1. User Super User must be able to manage database records:
   1.1. Insert records;
   1.2. Update records;
   1.3. Delete records.

US-3a-4.

As an Authenticated Super User I want to be able to manage Celery Worker tasks so I can better assess and manage operation.

Acceptance Criteria:

1. User Super User must be able to manage tasks:
   1.1. Add task;
   1.2. Change task;
   1.3. Delete task.

US-3a-5.

As an Authenticated Super User I want to be able to manage Celery Beat periodic tasks so I can better assess and manage operation schedule.

Acceptance Criteria:

1. User Super User must be able to manage periodic tasks:
   1.1. Add periodic task;
   1.2. Change periodic task;
   1.3. Delete periodic task.

US-3a-6.

As an Authenticated Super User I want to be able to manage Celery Workers so I can better assess and manage worker operation.

Acceptance Criteria:

1. User Super User must be able to manage periodic tasks:
   1.1. Add Worker;
   1.2. Change Worker;
   1.3. Delete Worker.

US-3a-7.

As an Authenticated Super User I want to be able to visualize task history so I can better assess whether or not the system is functioning correctly.

Acceptance Criteria:

1. Super User must be able to visualize task history:
   1.1. By area from the whole city to street level;
   1.2. By time from a yearly to a hourly time frame;
   1.3. By Traffic Density.
2. Super User must be able to drill up and down each of the described dimensions in the previous item.

US-3a-8.

As an Authenticated Super User I want to be able to Log out so I can successfully terminate my session and limitate access to unauthorized people from any machine using his account.

Acceptance Criteria:

1. Super User must be able to log out on any machine;
2. Super User must receive feedback from the system whether or not the session was successfully terminated;
3. Super User must be returned to the log in menu once log out has been successfully completed.

[4] Authenticated System Operator GUI

[4a] Administration Dashboard

US-4a-1.

As an Authenticated System Operator I want to be able to manage database tables so I can better assess and manage stored information.

Acceptance Criteria:

1. User System Operator must be able to manage database records:
   1.1. Insert records;
   1.2. Update records;
   1.3. Delete records.

US-4a-2.

As an Authenticated System Operator I want to be able to manage Celery Worker tasks so I can better assess and manage operation.

Acceptance Criteria:

1. User System Operator must be able to manage tasks:
   1.1. Add task;
   1.2. Change task;
   1.3. Delete task.

US-4a-3.

As an Authenticated System Operator I want to be able to manage Celery Beat periodic tasks so I can better assess and manage operation schedule.

Acceptance Criteria:

1. User System Operator must be able to manage periodic tasks:
   1.1. Add periodic task;
   1.2. Change periodic task;
   1.3. Delete periodic task.

US-4a-4.

As an Authenticated System Operator I want to be able to manage Celery Workers so I can better assess and manage worker operation.

Acceptance Criteria:

1. User System Operator must be able to manage periodic tasks:
   1.1. Add Worker;
   1.2. Change Worker;
   1.3. Delete Worker.

US-4a-5.

As an Authenticated System Operator I want to be able to visualize task history so I can better assess whether or not the system is functioning correctly.

Acceptance Criteria:

1. System Operator must be able to visualize task history:
   1.1. By area from the whole city to street level;
   1.2. By time from a yearly to a hourly time frame;
   1.3. By Traffic Density.
2. System Operator must be able to drill up and down each of the described dimensions in the previous item.

US-3a-6.

As an Authenticated System Operator I want to be able to Log out so I can successfully terminate my session and limitate access to unauthorized people from any machine using his account.

Acceptance Criteria:

1. System Operator must be able to log out on any machine;
2. System Operator must receive feedback from the system whether or not the session was successfully terminated;
3. System Operator must be returned to the log in menu once log out has been successfully completed.

Appendix C

Quality Requirements Scenarios - Utility Tree

The quality requirements that were elicited in subsection 4.3.3 did not give us much information. To understand how these quality might affect our system, we further refined them into scenarios that give use additional information about how the system to be built should operate.

Although constructing these scenarios helped us understand how quality requirements affected system operation, having a lot of scenarios could severely hinder architectural construction as they did not have the same importance and were often contradictory.

Therefore, it was important to prioritize this scenarios. Since each stakeholder had its personal opinion and priorities it we had to reach some kind of compromise.

In order o reach such compromise, each stakeholder should classify each scenario according to two different dimensions:

- **Architectural Impact [H- High, M- Medium, L- Low]:** The architectural impact this scenario will have in the architecture. Scenarios that have high impact should drive the architectural choice;
- **Value to Business [H- High, M- Medium, L- Low]:** Defines how critical this scenario is for business. Scenarios that have high value to business should drive architectural choice.

By voting and compromising, stakeholders defined which scenarios were more important and, in case scenarios were contradictory, which should be preferred and which should be left out of the system.

Quality Requirement	Requirement Refinement	Scenarios
Extendability	Adding a new product	Ubiwhere wants to create an area desirability product targeting the real estate market. System architecture must be able to incorporate this new feature in a single development cycle of one month. **(H, M)**
	Adding new Features	Ubiwhere wants to add an automated map loader to the system. System architecture must be able to incorporate this new feature in a single development cycle of one month. **(H, M)**
Interoperability	Documented exchange and file formats; Open communication protocols	Ubiwhere wants to feed data to our system from a brand new or existing sensor aggregator.System architecture must be able to allow our system to connect and receive data from the sensor aggregator without any major modification to our system. **(H, H)**
		Ubiwhere wants a Emergency Response System to use data collected from our system and stored in the OpenGIS database. System architecture must be able to allow our system's database to connect and feed data to the ERS without any major modification to our system. **(M, M)**
		Ubiwhere wants to incorporate web map routing into a travel website. System architecture must allow our system's routing service, geocoding service and web map service to provide services to another website without any major modification to our system. **(M, H)**
		Ubiwhere wants a Emergency Response System to use the maps compiled by our system in its routing engine. System architecture must allow our file repository service to provide services to another website without any major modification to our system. **(M, H)**

Modularity	Flexibility to replace components	Ubiwhere wants to replace the the sensor frontend server module by a more complex one that allows HTTPS to be used. System architecture must allow the server to be replaced without any major modification to the rest of the system. **(H, M)**
	Flexibility to replace modules	Ubiwhere wants to replace the module that validates sensor data with a more complex one that detects data tampering. System architecture must allow this module to be replaced without any major modification to the rest of the component. **(H, M)**
Portability	System Deployment	Ubiwhere wants to sell the system to a client whose web servers use Windows Technology. The system architecture must allow the system to be deployed in any major operating system without any modifications to it. **(H, H)**
Reusability	Use components in other systems	Ubiwhere wants to use the map compiler component to be used in a Emergency Response System it is developing. Component architecture must allow this component to be used in another system without any major modification to it. **(H, H)**
	Use the same module in several components of the system and other systems	Ubiwhere wants to use the same logging module in several components of our system. Module architecture must allow modules to be reused along the system without any modification to the components. **(H, L)**
Scalability	Growing the System	Ubiwhere wants to connect the system to a sensor aggregator that will create ten times the number of sensor readings the system processes in normal operation. The system must be able to autonomously accommodate such a growth in sensor readings without any major modification to it. **(H, H)**

		A publicity campaign has resulted in a sudden peak of user visits to the web page. The system is experiencing three times the normal number of requests. The system must be able to autonomously accommodate such a growth client visits without any major modification to it. **(H, H)**
		The end of a contract several metropolitan areas has resulted in the number of sensor inputs to be reduced by ten times the number of reads it processes in normal operations. The system must be able to autonomously accommodate such a diminution in sensor readings without any major modification to it. **(H, H)**
		A natural disaster has resulted in communication outage in a certain area. The number of clients visiting the web site has diminished by 90%. The system must be able to autonomously accommodate such a diminution in client visits without any major modification to it; **(H, H)**
Security	Control Access	An unregistered user wants to access the system without registering and logging in first. The system does not allow an unregistered user to access any information without the user registering and logging in first; **(M, L)**
		A registered user wants to access the system without logging in first. The system does not allow a registered user to access any information without the user registering and logging in first; **(M, L)**
	Integrity	A third party wants to compromise the system's integrity by performing SQL injection in the message payload that comes from the sensor aggregator. The system will be able to filter and validate each message it receives, dropping any message that contains out of parameters information **(M, L)**

		A third party wants to disseminate malicious software by injecting a vulnerable form with some malicious JavaScript. The system will be able to identify and filter any type of Cross-site Scripting attacks. **(M, L)**
Usability	Proficiency training	Ubiwhere wants to make the system available to the general public as a web browser page. The system must allow a non proficient user to become proficient with the system in less than an hour. **(L, M)**
	Normal Operations	Ubiwhere wants to make the system available to the general public as a web browser page. The system must allow a proficient user to complete any task without the system introducing any delays. **(L, M)**

Table C.1: Quality Requirements Scenarios - Utility Tree

Inside the parenthesis the first value is the architectural impact and he second its value to business.

Appendix D

Initial Architecture

D.1 System Decomposition

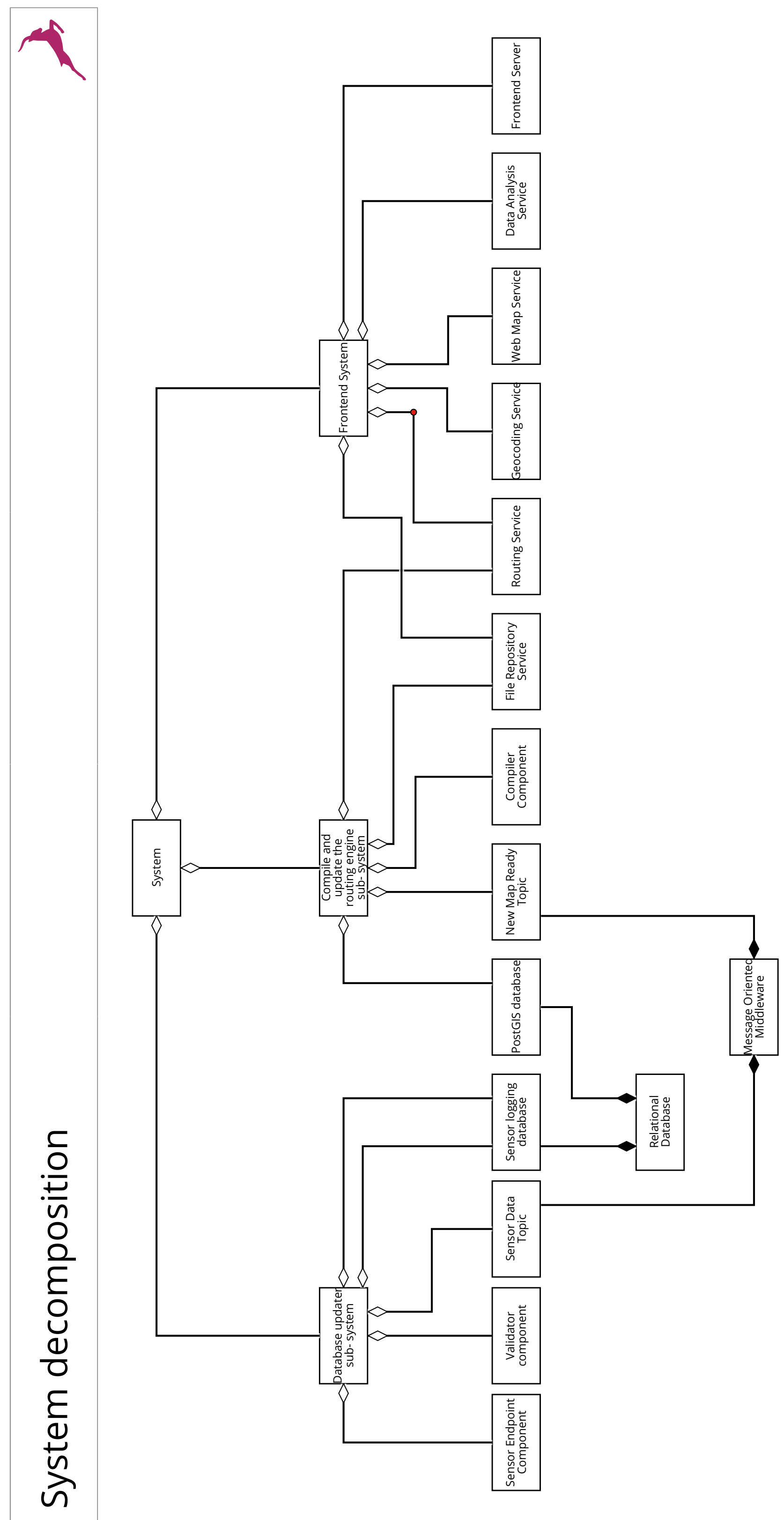


Figure D.1: System decomposition

D.2 Data Pipe Flow

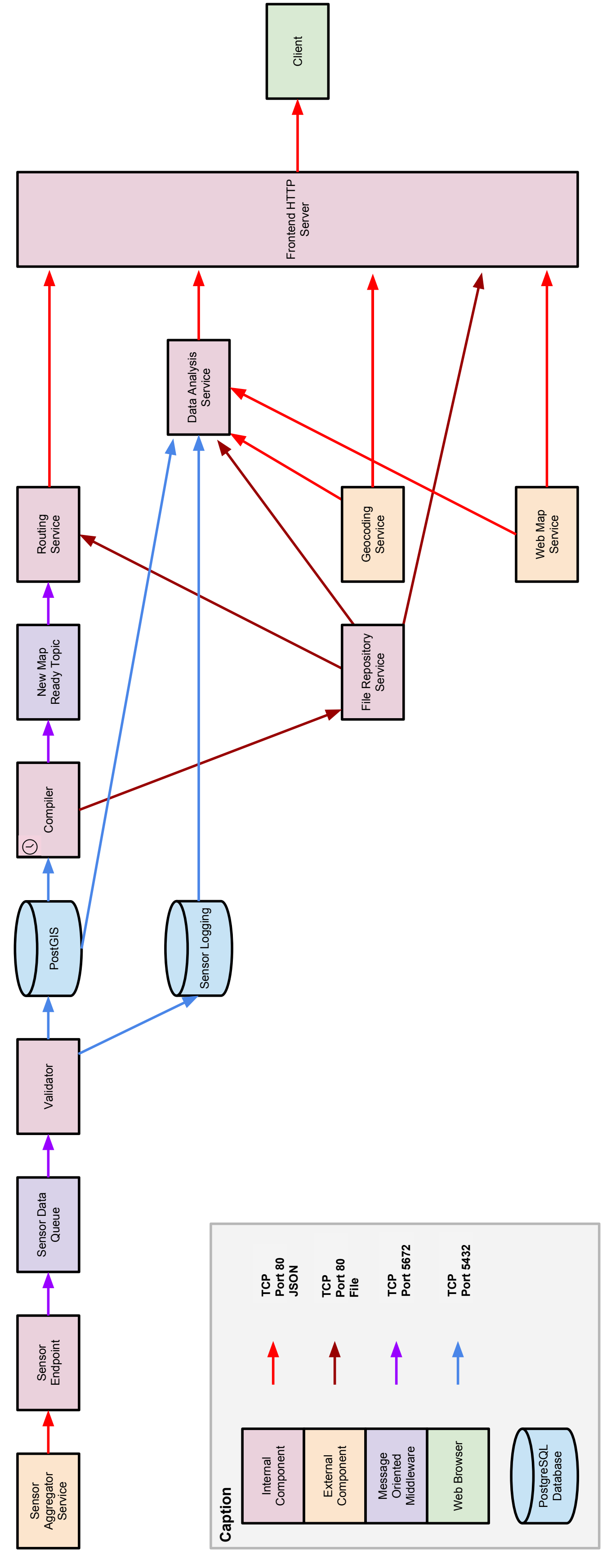


Figure D.2: Data Pipe Flow

D.3 Business Processes

D.3.1 Database Updater System

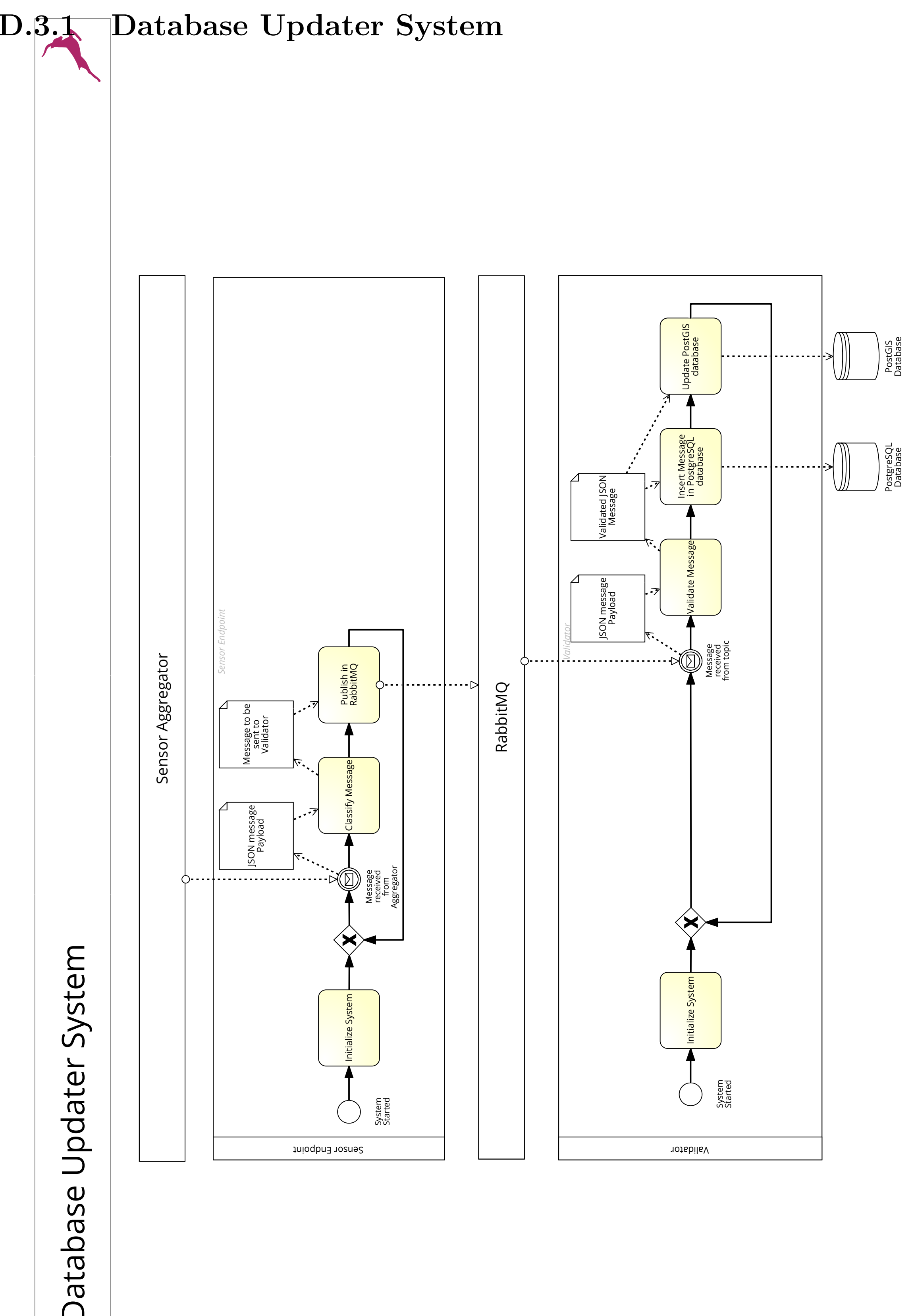


Figure D.3: Database Updater System

D.3.2 Compile and Update Routing Engine System

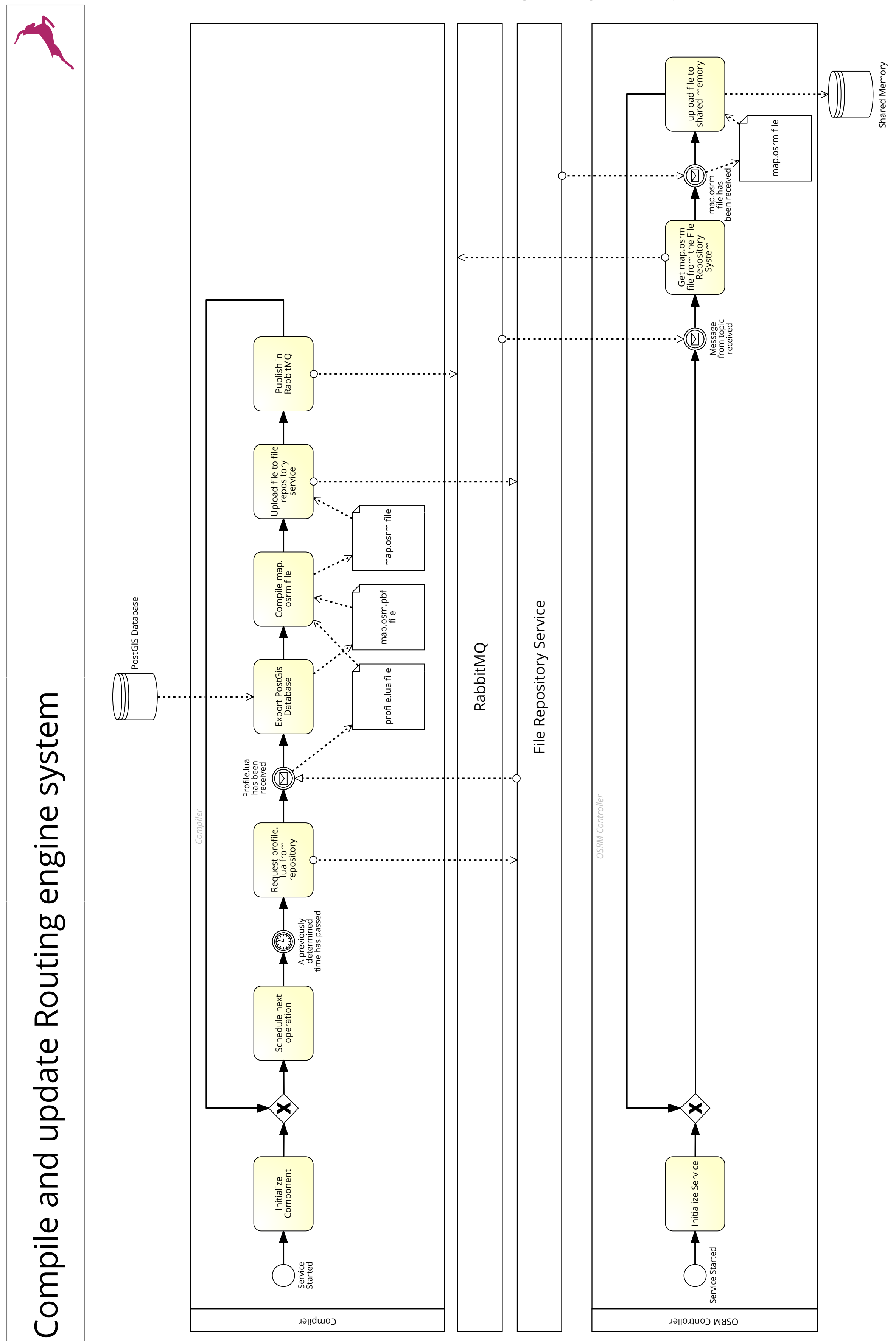


Figure D.4: Compile and Update Routing Engine System

D.3.3 Sensor Endpoint

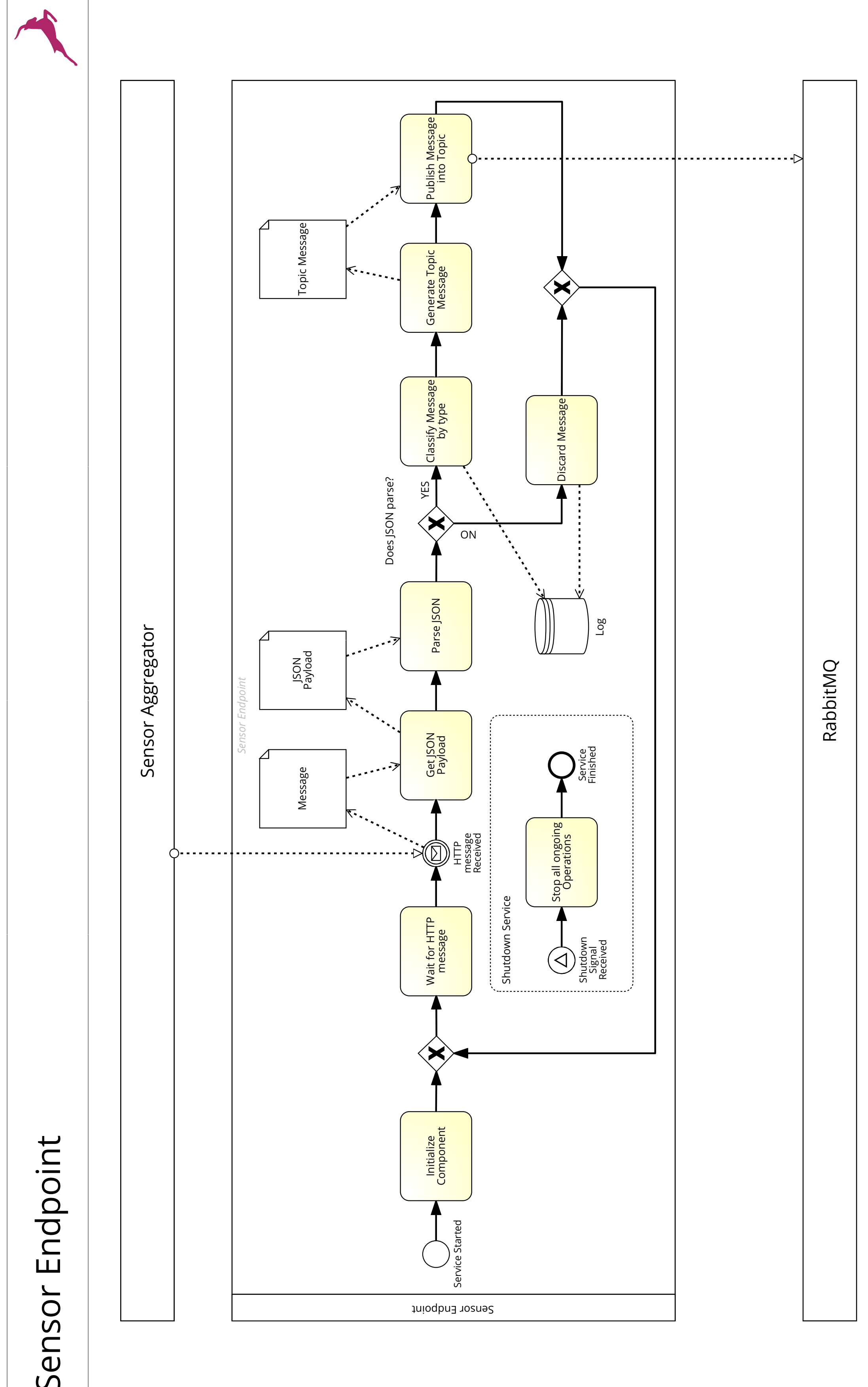


Figure D.5: Sensor Endpoint

D.3.4 Validate

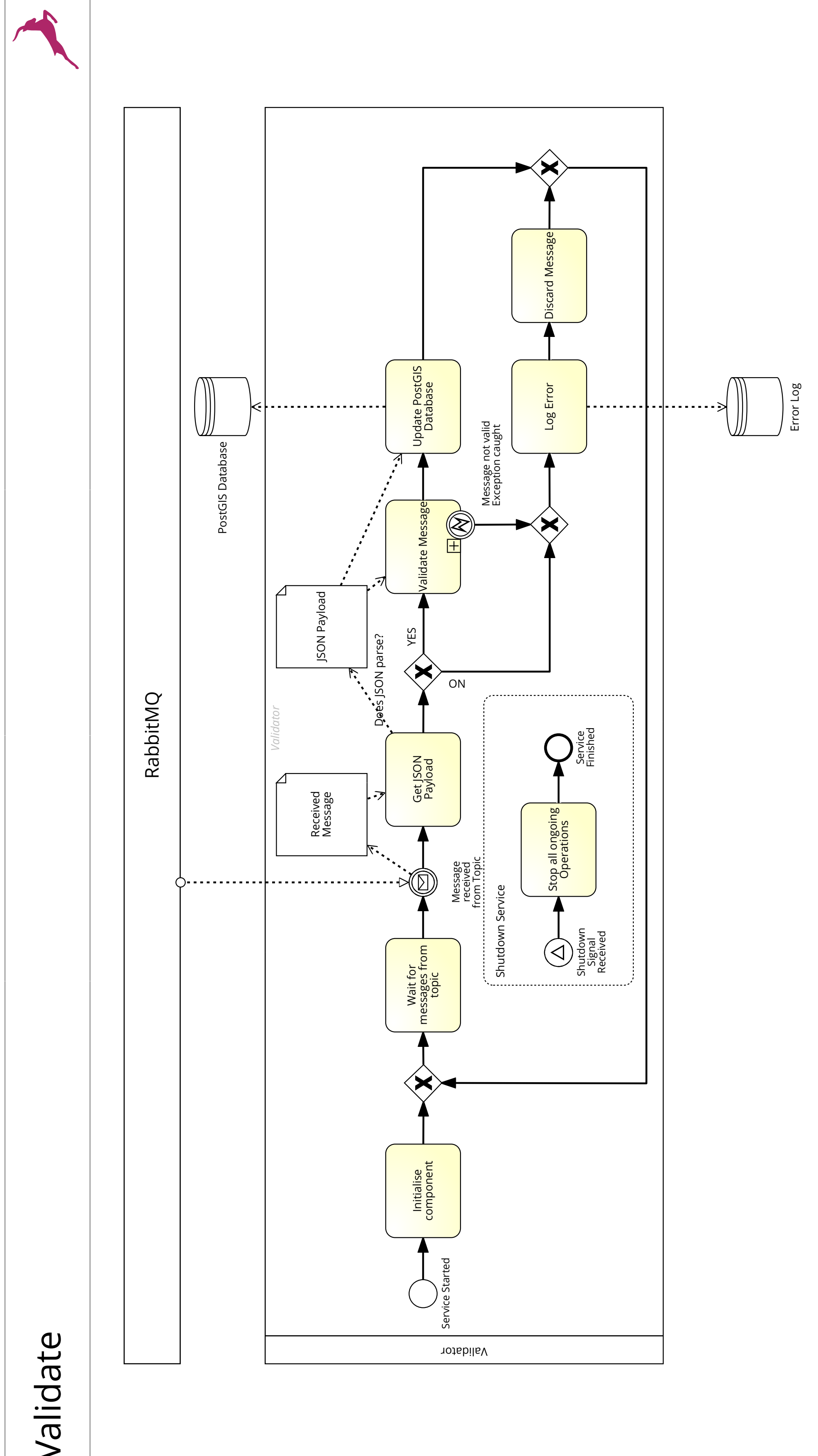


Figure D.6: Validate

D.3.5 Validate Message

Validate Message

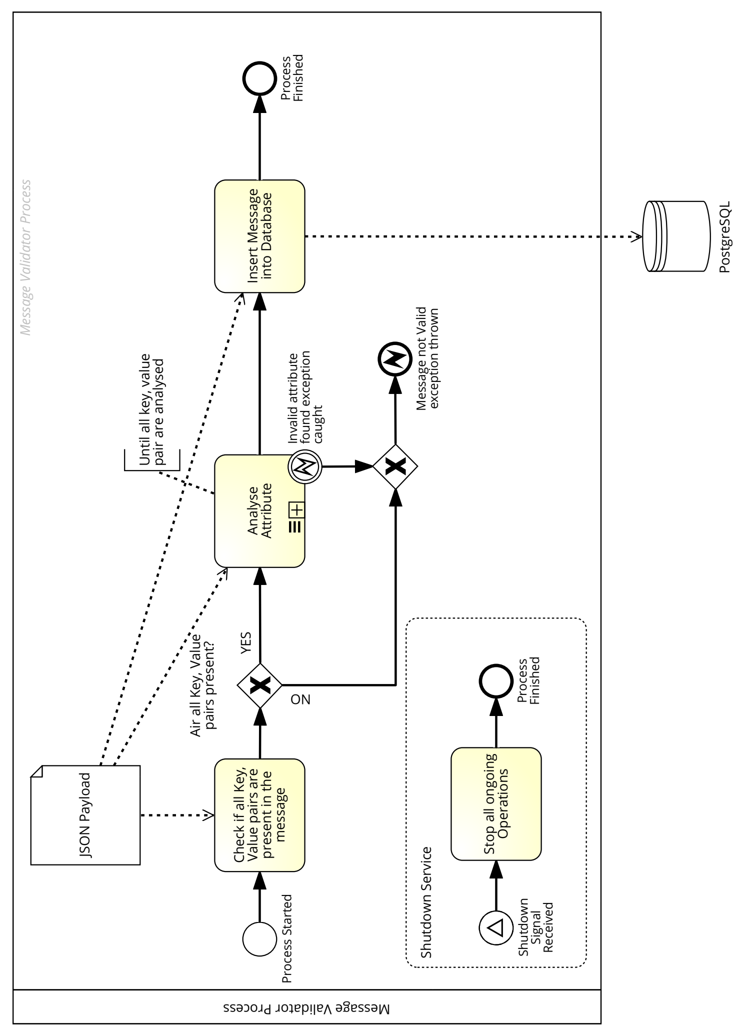


Figure D.7: Validate Message

D.3.6 Analyse Attribute

Analyse Attribute

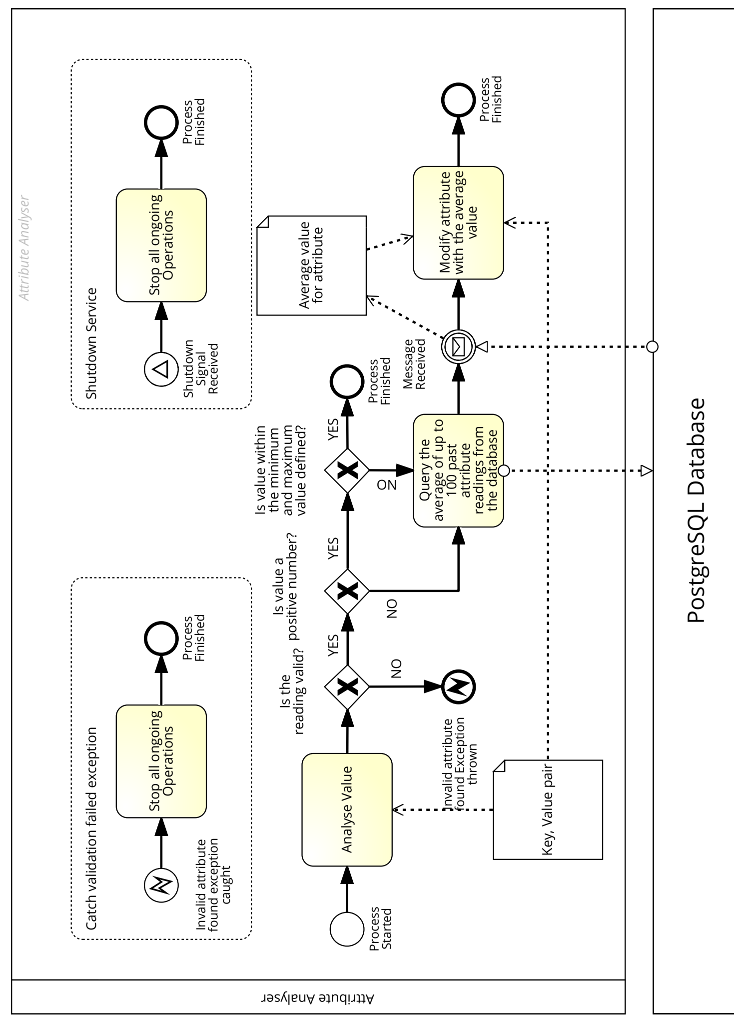
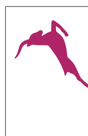


Figure D.8: Analyse Attribute

D.3.7 Compile OSRM File



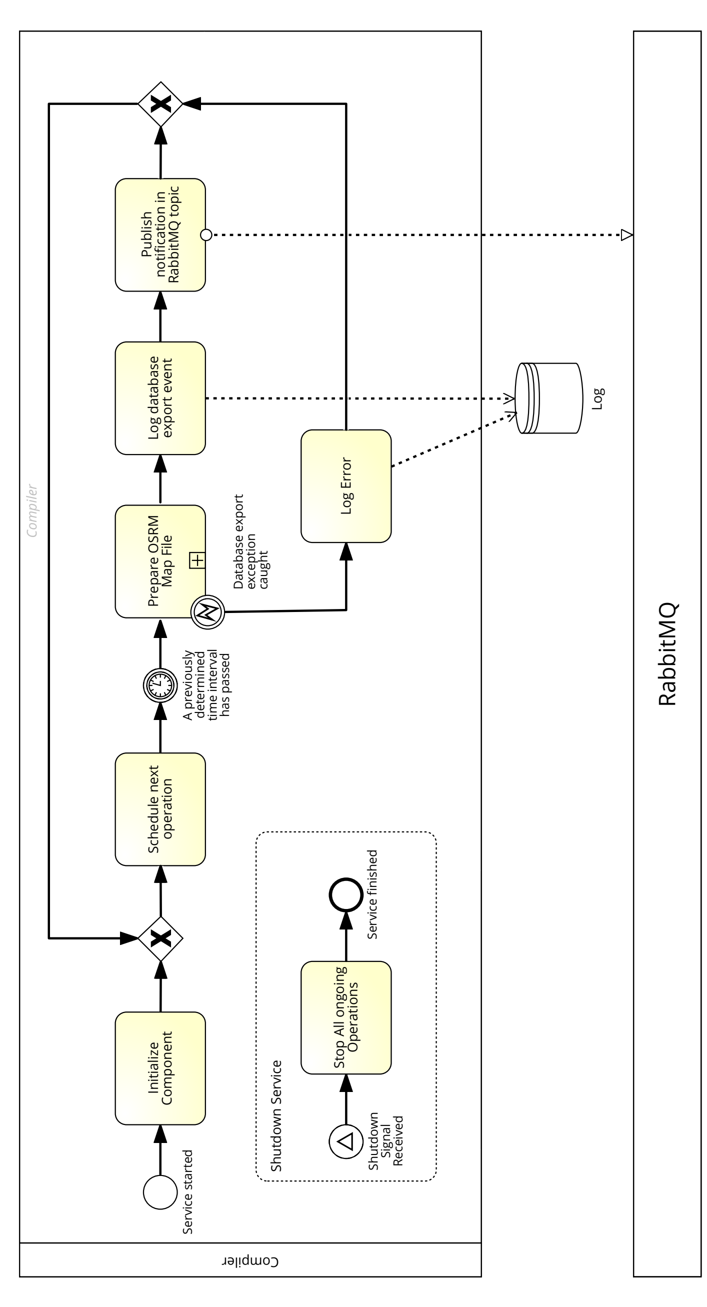


Figure D.9: Compile OSRM file

D.3.8 Prepare OSRM Map File

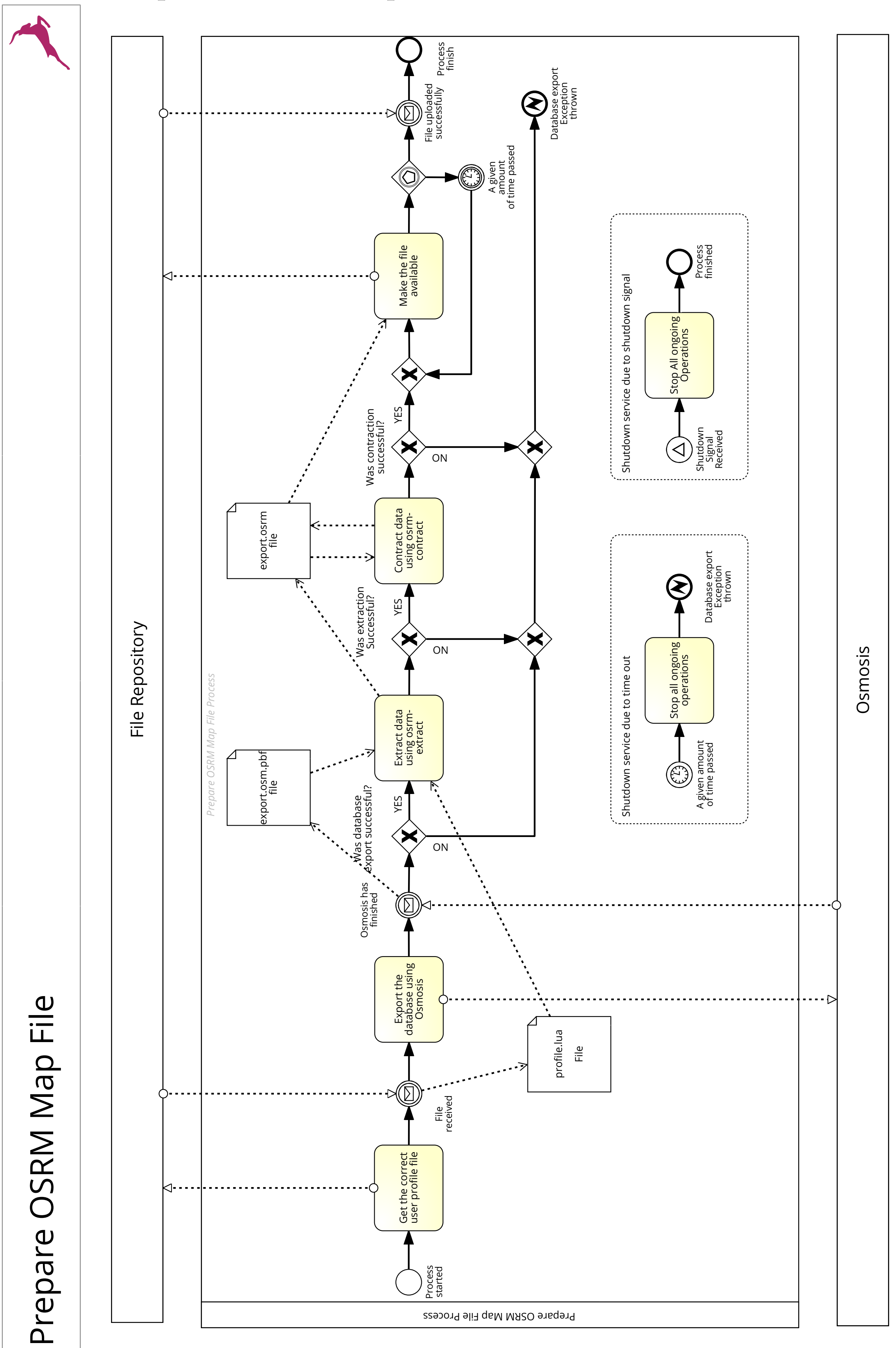


Figure D.10: Prepare OSRM Map File

D.3.9 Make Files Available

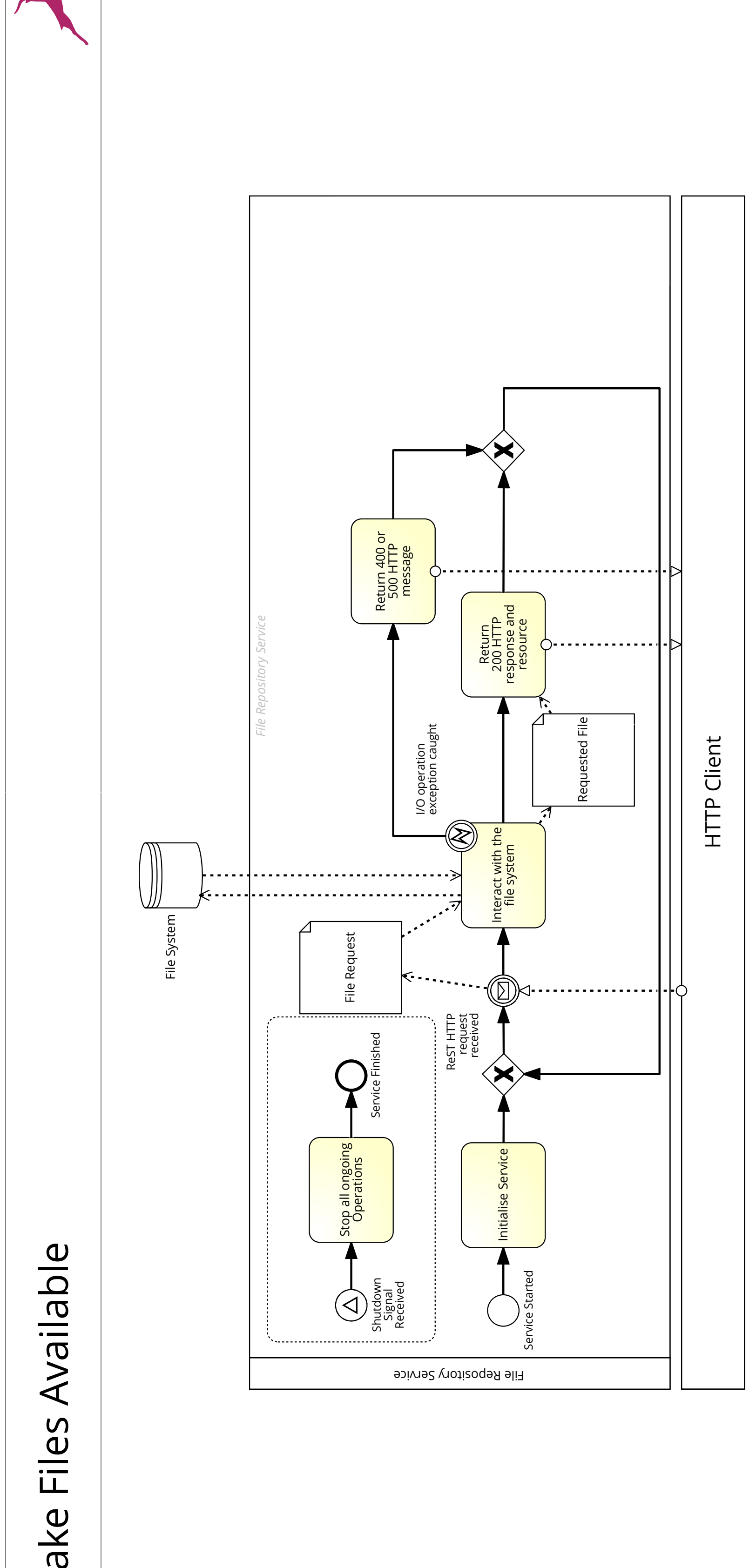


Figure D.11: Make Files Available

D.3.10 Load New Map File to Router

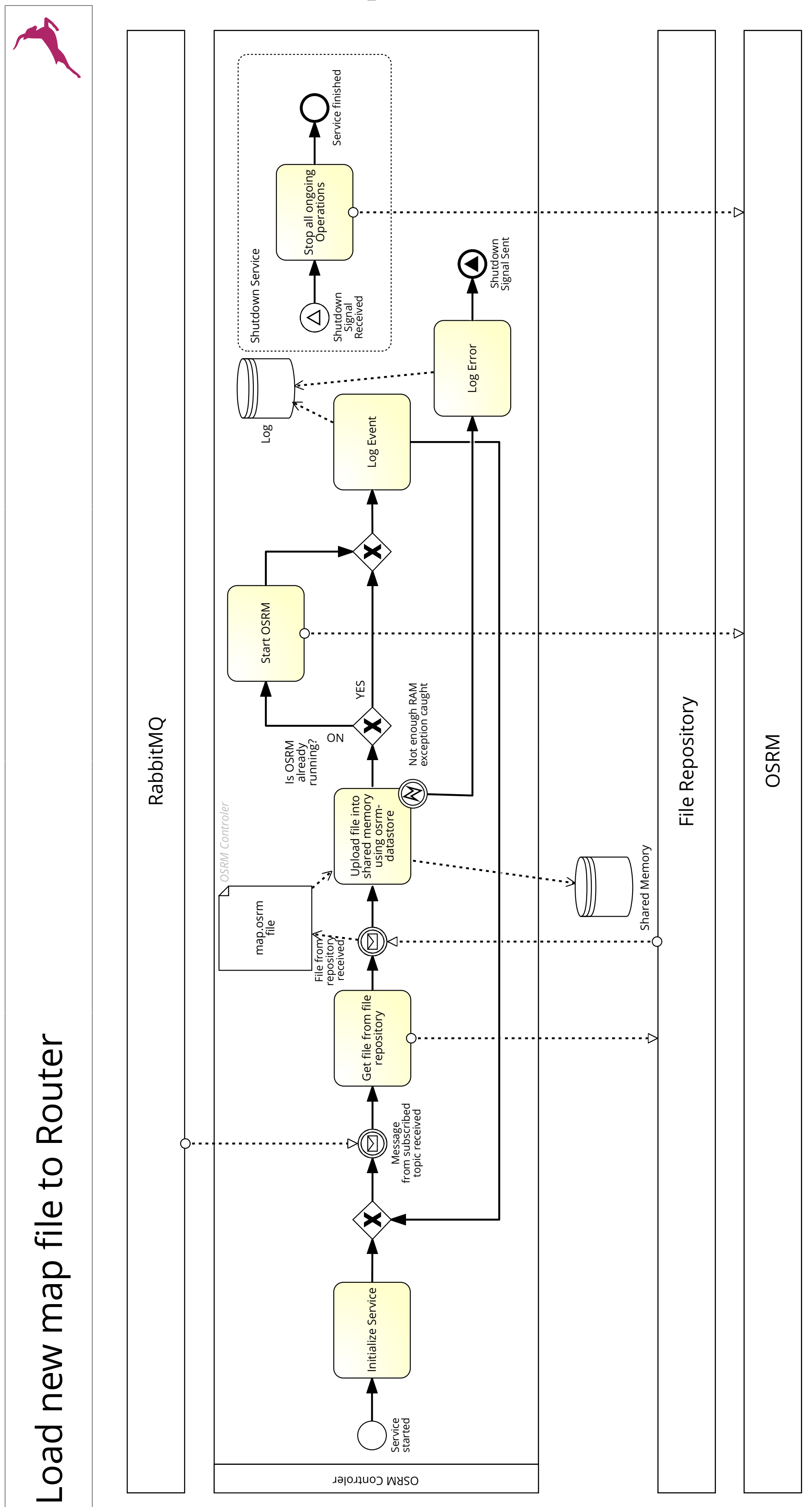


Figure D.12: Load New Map File to Router

D.3.11 System Architecture - Layer View

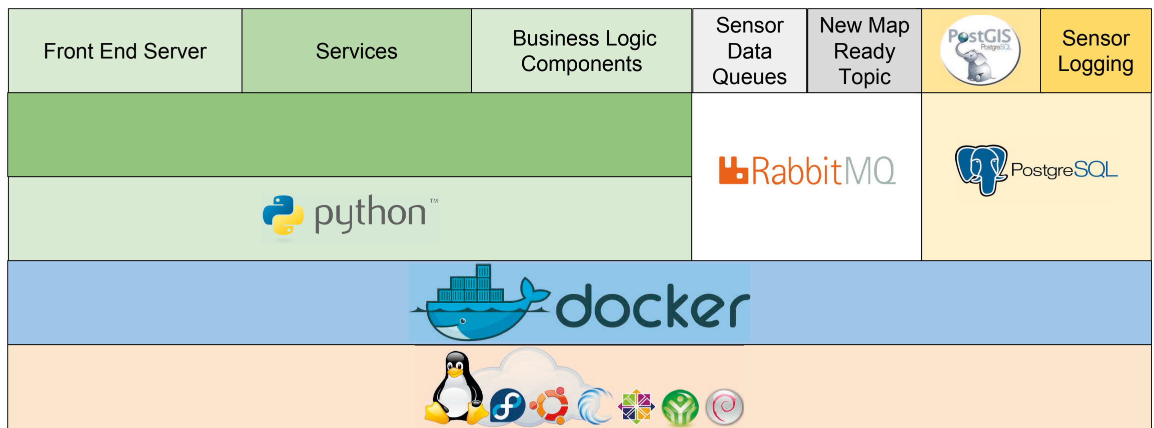
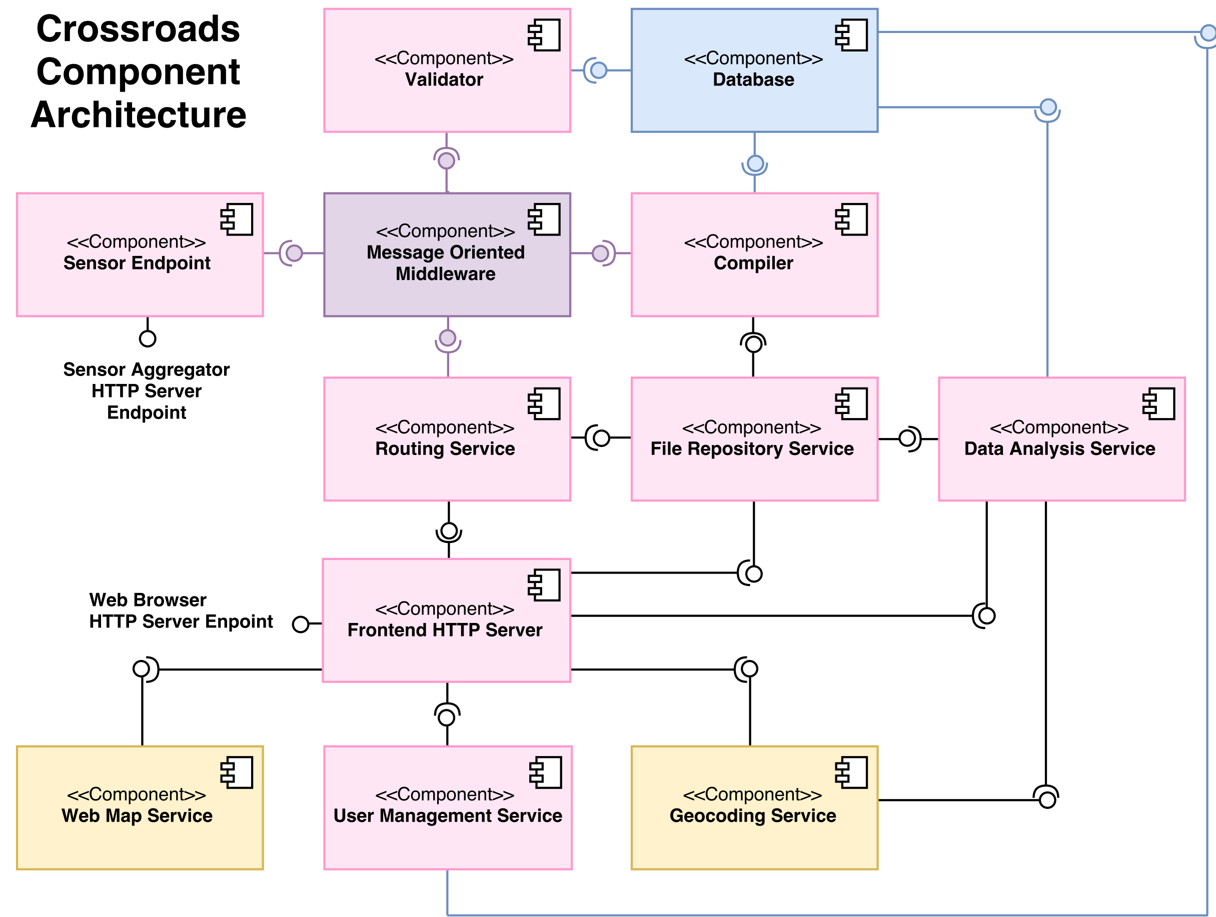


Figure D.13: System Architecture - Layer View

D.3.12 System Architecture - Component Diagram



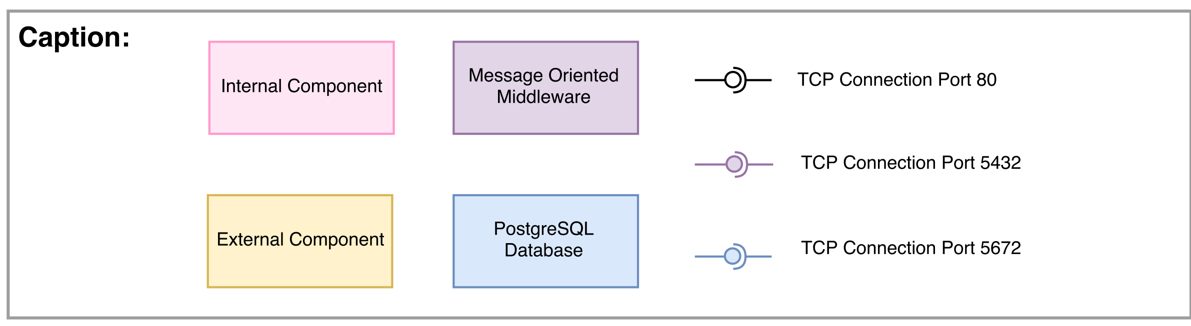


Figure D.14: System Architecture - Component Diagram

D.3.13 System Architecture - Components Architecture Diagram 1

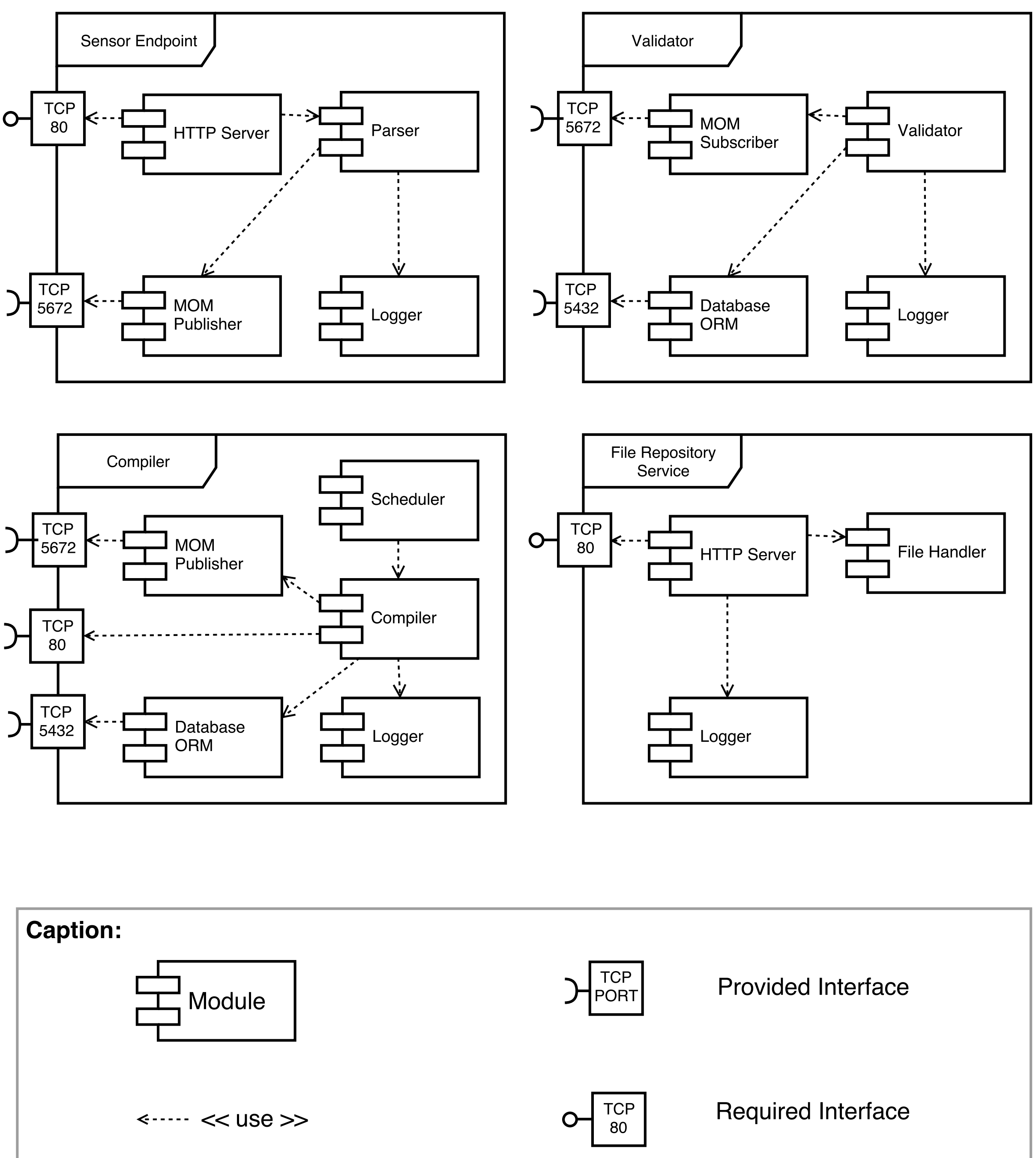


Figure D.15: System Architecture - Components Architecture Diagram 1

D.3.14 System Architecture - Components Architecture Diagram 2

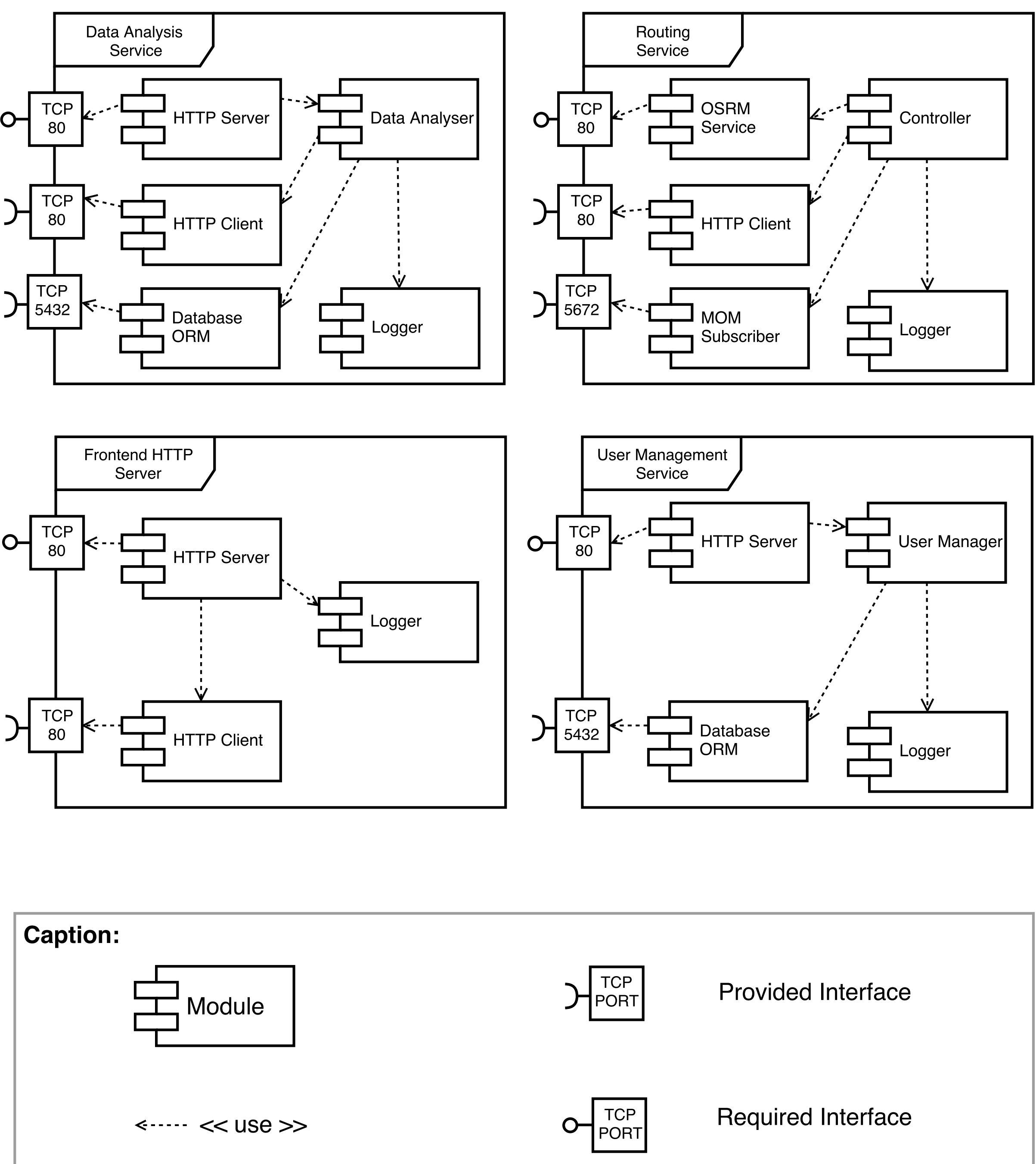


Figure D.16: System Architecture - Components Architecture Diagram 2

Appendix E

Risk Analysis

E.1 Purpose

This section has the objective of assessing risk and dealing with the degree of uncertainty such an exploratory process as this project comprehends.

Objectives defined for the project did not give any objective information about what needed to be achieved to consider the project a success. To measure success, we defined criteria and metrics that constituted the threshold of success for the project. Failure to achieve any of these criteria would lead to project failure.

At this point, many factors may pose obstacles to these minimum objectives being met. This document functions as a tool to allow better control of the whole process, to better identify these threats.

Sources, conditions, and consequences were elicited and analyzed by the stakeholders, to determine their impact, probability, and timeframe.

To deal with these risks, we placed them in a risk matrix. Stakeholders voted to order them importance. With risks prioritized, we devised a risk mitigation plan.

E.1.1 Threshold of Success

To be able to assess if the project had reached its goals and better evaluate the development process, we proposed several metrics and criteria that constituted the threshold of success for our project. Failure to achieve any of these goals immediately lead to the project to be deemed unsuccessful.

Threshold of Success:

- The Must Have User Stories as defined in section 4.3.1 of this document are developed and delivered by June 2017;
- The system respects all Constraints as defined in section 4.3.2 of this document;
- The system respects all Non Functional Requirements as defined in section 4.3.3 of this document;

- The workload is well distributed and tasks completed within the 1176 hours (42 ECTS) allocated to the internship;
- The process respects the proposed High-level Plan and Milestones with a less than 2 weeks discrepancy.

These criteria defined our threshold of success. Failure to achieve any of them should lead to project failure.

E.1.2 Risk Identification

We identified potential threats to the project to diminish the uncertainty associated with such an exploratory process. Risk identification, analysis, and mitigation were a priority.

Then we classified in terms of probability of occurring, their overall impact on the project and when they could occur.

Description	Low	Medium	High
Probability	<33.33%	33.33% to 66.66%	>66.66%

Table E.1: Risk Probability

Impact	Description
Marginal	The threshold of success is not compromised. Issues associated with this risk can be easily dealt with.
Critical	The threshold of success can still be achieved at great effort or cost.
Catastrophic	The threshold of Success cannot be achieved.

Table E.2: Risk Impact

Time Frame	Description
Short Term	Risk can occur in the next few weeks.
Mid Term	Risk can occur in between one and three months.
Long Term	Risk can occur in more than three months.

Table E.3: Risk Time Frame

E.2 Iteration One

E.2.1 Risk Identification

Risk 1: Lack of Complete Sensor Infrastructure or Data sets

Since the beginning of the project, the lack of a complete data was a major problem. This fact posed a major threat to the project. Without quality data, it would be hard to evaluate our system.

Since this project was not meant to production, a great deal of information could be obtained using incomplete data or generating our own data set. Nevertheless, it would be impossible to validate the system in the real world.

To mitigate this risk, We should keep looking for complete data sets and try to validate our concept against it. We should also develop the necessary tools to help us visualize data.

Risk 1	Description
Source	Lack of complete traffic and air quality sensor infrastructure and data sets.
Condition	Uncertainty in requirements elicitation and architecture definition
Consequence	Requirements and Architecture might prove inadequate for the project and the system may be impossible to validate
Probability	High
Impact	Critical
Time Frame	Short Term
Mitigation Plan	Look for an adequate data set, If we cannot find one, we may have to generate it

Table E.4: Risk 1

Risk2: Fulfilling our GIS data necessities using crowd funded map data

At this point, we had no real alternative to use OSM map data for our project. Having completely free map export, built using contributions of a large community, although it produced highly accurate maps, was not without its challenges.

After analyzing map data imported into a PostGIS database, we identified lots of overlapping information, inaccuracies, and missing tags. This data would have to be extracted and normalized.

Since we would initially be using external web map and geocoding services, We could not guaranty all the system's services were working with the same map version.

To mitigate this risk, we should devise a system that operated with a single version of a highly normalized map. Extracted information should not contain, discrepancies, missing fields and should not be duplicated.

Risk 2	Description
Source	Fulfilling our GIS data necessities with crowd sourced map data creates difficulties in map data usage
Condition	Data inaccuracy, missing tags in elements, lack of data validation
Consequence	May cause delays in system development and the system may be impossible to validate
Probability	High
Impact	Critical
Time Frame	Medium Term
Mitigation Plan	Remove irrelevant data and conform it to some hard standard

Table E.5: Risk 2

Risk 3: Steep Learning Curve of some of the technologies

The system to be built was quite complex. There would not be enough time to fully study Libraries and service APIs. Although we had some experience with Python, the Django Web Frameworks was completely new to us. We would also need to learn how to properly configure Docker and all the other containers that would be included in the project.

Having so many different new technologies to learn in such a short period, could cause some delays in development. There would have to be some flexibility in terms of the time frame in order to accommodate the necessary time to develop the necessary skills.

Risk 3	Description
Source	Steep learning curve of some of the technologies to be used
Condition	Too much time spent acquiring the necessary skills
Consequence	May cause delays in development and requirement unfulfillment
Probability	Low
Impact	Critical
Time Frame	Medium Term
Mitigation Plan	Adjust the development time frame in order to have enough time to learn the necessary skills

Table E.6: Risk 3

Risk 4: Change Accommodation

At this point, there was a great degree of uncertainty with the project. As new information became available, it would be necessary to make changes to requirements and architecture. It would be necessary to accommodate these changes to the project.

We should have regular meetings with the stakeholders, to inform them of progress and have a process in place to be able to operate the necessary changes to requirements and architecture.

Risk 4	Description
Source	Necessity to make changes to the project's requirements/ Architecture
Condition	Requirements do not meet the client's necessities or the architecture is inadequate to meet requirements
Consequence	There might be delays in development or requirement unfufillment
Probability	Medium
Impact	Critical
Time Frame	Medium Term
Mitigation Plan	Have regular meetings with the product owner and a change management process in place

Table E.7: Risk 4

Risk 5: Non Acceptance

By the end of the project, the developed software should meet all functional and operational requirements as well as respect all elicited restrictions.If it was not up to expectation, there was a possibility Ubiwere would not accept it and the project would fail.

To meet the stakeholder's expectations, we should have regular meetings with them and try to negotiate a realistic high-level plan and time frame- We should also leave room to accommodate possible changes to the requirements that the stakeholders deem necessary as long as they would not diverge too greatly from the project scope and time frame.

Risk 5	Description
Source	Non acceptance by the product owner of the developed software
Condition	Software does not meet requirements and restrictions or meet the deadline
Consequence	May cause project failure
Probability	Low
Impact	Catastrophic
Time Frame	Long Term
Mitigation Plan	Have regular meetings with the client and negotiate a realistic High-level Plan and Time frame.

Table E.8: Risk 5

E.2.2 Risk Prioritization

To prioritize risk and determine which ones were more urgent to address, we used a Risk Exposure Matrix to figure out what was the type of exposition risk was to be expected. After determining exposition risk, we sorted them by exposition risk and timeframe. With this information, we prioritized risks. With this information, we had an idea which issues should be addressed first to act proactivity and be able to mitigate them before they even appear.

	Likelihood	Low	Medium	High
Impact				
Catastrophic		Risk 5		
Critical		Risk 3	Risk 4	Risk 1, 2
Marginal				

Table E.9: Risk Exposure Matrix Iteration 1

Exposition Risk	Low	Medium	High	Critical

Table E.10: Exposition to Risk Iteration 1

ID	Exposition Risk	Time Frame
1	High	Short Term
2	High	Medium Term
4	Medium	Medium Term
5	Medium	Long Term
3	Low	Medium Term

Table E.11: Risk Prioritization Iteration 1

E.2.3 Risk Mitigation Plan

After analyzing table E.11, it became clear that the lack of complete traffic data should be dealt with first. To evaluate traffic data adequacy, we should develop the necessary infrastructure to better visualize sensor coverage and data quality. An initial system

that could transform received data into a GeoJSON object and could be layered on a browser-based web map would help us assess sensor coverage and data quality.

With this tool, we should be able to select an adequate traffic data source. This source would allow us to assess if our architecture was adequate to transform its data into useful information and proceed with our project.

E.3 Iteration 2

E.3.1 Overall Risk Evolution

By developing the necessary infrastructure, we were able to find a data source that proved adequate for the initial prototype. This took care of the most immediate risk for the time being.

During this iteration, we would need to address the risks related to our map source adequacy. An initial analysis of the OSM map data contained in a PosGIS database had shown that it contained duplicates and discrepancies. It would hard to use data as it was. The possibility of exporting the database, on the fly, had proven impossible for large maps.

We would have to study alternatives that could allow us to incorporate traffic information into the map used by the routing engine.

E.3.2 Threshold of Success

At this point the threshold of success for our project remained the same.

E.3.3 Existing Risks Evolution

Lack of Complete Sensor Infrastructure or Data sets

With a data source chosen, we would be able to start developing our system. Even though the situation was not ideal, we should able to obtain valuable information that we could use when a better data source could be found.

Risk 1	Description
Source	Lack of complete traffic and air quality sensor infrastructure and data sets.
Condition	Uncertainty in requirements elicitation and architecture definition
Consequence	Requirements and Architecture might prove inadequate for the project and the system may be impossible to validate
Probability	Medium
Impact	Critical
Time Frame	Medium Term
Mitigation Plan	Look for an adequate data set, If we cannot find one, we may have to generate it

Table E.12: Risk 1 Evolution Iteration 2

E.3.4 New Risks Identification

Risk 6: Exporting and extracting the complete map form the PostGIS database

After initial testing, it became clear that we would not be able to export the PostGIS database on the fly. We needed to find a viable alternative. After some research, we discovered the OSRM experimental traffic feature. This feature allowed us to integrate traffic data into OSRM without the periodically exporting the complete database. Since the feature was still experimental, we would have to evaluate if it produced the necessary results.

Risk 6	Description
Source	Exporting and extracting the complete map from the PosGIS database
Condition	Exporting and extracting a large map is a too time consuming task.
Consequence	The system may not be able to deal with large maps
Probability	High
Impact	Catastrophic
Time Frame	Short Term
Mitigation Plan	Assess whether or not the OSRM experimental traffic feature can be used to incorporate traffic data into the compressed web map

Table E.13: Risk 6

E.3.5 Risk Prioritization

	Likelihood	Low	Medium	High
Impact				
Catastrophic		Risk 5		Risk 6
Critical		Risk 3	Risk 1, 4	Risk 2
Marginal				

Table E.14: Risk Exposure Matrix Iteration 2

Exposition Risk	Low	Medium	High	Critical

Table E.15: Exposition to Risk Iteration 2

ID	Exposition Risk	Time Frame
6	Critical	Short Term
2	High	Short Term
1	Medium	Medium Term
4	Medium	Medium Term
5	Medium	Long Term
3	Low	Medium Term

Table E.16: Risk Prioritization iteration 2

E.3.6 Risk Mitigation Plan

After prioritizing risk, it became clear that the inability of exporting large maps on the fly was a problem. We would have to deal wit it immediately.

The OSRM experimental traffic feature looked promising. Since it was experimental, we would have to test it. If it worked properly it would mitigate with the two most important risk.

In order to assess whether or not this feature could be used we devised to main activities for this iteration:

1. Devise a way to convert geographic coordinates obtained from sensor data into the corresponding *osm_id* Nodes. This operation would produce the traffic data necessary for the OSRM experimental traffic feature;

2. Conduct a series experiments to assess if whether or not the OSRM experimental traffic feature was suitable for our project.

E.4 Iteration 3

E.4.1 Risk Evolution

In the last iteration, we conducted a series of experiments that allowed us to conclude that the experimental OSRM traffic feature would be a much more attractive alternative to introducing traffic information into our routing engine.

To properly integrate this feature in our system, we needed to convert point coordinates arrays into *osm_id* nodes. We used the geocoding and the OSM map service to achieve this goal. Even though we were getting the necessary Nodes, we did not know if these elements were present in the OSRM map. If we were feeding the routing engine useless information, the system would not perform adequately. We would have to deal with this new issue.

E.4.2 Threshold of Success

The threshold of success for our project remained the same.

E.4.3 Existing Risks Evolution

Exporting and extracting the complete map from the PostGIS database

Since we were no longer using the PostGIS database as our map source, this risk no longer existed.

E.4.4 New Risk Identification

As a System Using nodes obtained from the OSM Web Map API with the OSRM experimental traffic feature

We had no guaranty that the *osm_id* Nodes obtained from the web map server API existed in the compressed OSRM produced map graph. This issue was a major risk to the project and had to be dealt with as soon as possible.

Also, since we were using only the middle coordinate of the line-string object that constituted each path, we had no guaranty the *osm_id* Way returned by the OSM server API covered the whole path.

Risk 7	Description
Source	Using nodes obtained from the OSM Web Map API with the OSRM experimental traffic feature
Condition	No guaranties that nodes are present in OSRM compressed map graph
Consequence	The system may not operate properly
Probability	High
Impact	Critical
Time Frame	Short Term
Mitigation Plan	Assess whether or not nodes are present. Find alternatives to using the geocoding service to match paths to *osm_id*

Table E.17: Risk 7

Risk 8: The system requires validation

The final objective of this iteration was to produce a complete initial prototype. This initial prototype should be validated, otherwise, it would be worthless.

Since we could not perform real world validation, we would have to test our prototype against existing alternatives. This test would not validate our prototype but would give some kind of feedback whether or not the system was working properly.

Our system shared no components with GraphHopper traffic data integration demonstration. if they returned a similar response to a request that featured the same source and target coordinates, it would be a strong indicator that our system was operating properly.

Risk 8	Description
Source	System requires validation
Condition	No guaranty data output reflects real world traffic contitions
Consequence	The system might be worthless
Probability	High
Impact	Critical
Time Frame	Short Term
Mitigation Plan	Compare our system's output with the GraphHopper traffic integration demonstration that used the same traffic data source.

Table E.18: Risk 8

E.4.5 Risk Prioritization

	Likelihood	Low	Medium	High
Impact				
Catastrophic		Risk 5		
Critical		Risk 3	Risk 1, 4	Risk 2, 7, 8
Marginal				

Table E.19: Risk Exposure Matrix Iteration 3

Exposition Risk	Low	Medium	High	Critical

Table E.20: Exposition to Risk Iteration 3

ID	Exposition Risk	Time Frame
8	High	Short Term
2	High	Short Term
7	High	Short Term
1	Medium	Medium Term
4	Medium	Medium Term
5	Medium	Long Term
3	Low	Medium Term

Table E.21: Risk Prioritization Iteration 3

E.4.6 Risk Mitigation Plan

For this iteration, three activities were planned to mitigate risk:

Layer the Node Arrays, obtained from the OSM service, on the browser-based web map

We had to guaranty that the array of Nodes, obtained through the reverse geocoding service and the OSM Map server API, represented the array of sensor coordinates from which the central point was being calculated.

We planned to translate the node array into GeoJSON objects that could be layered on a map using our already developed browser based frontend. By comparing the obtained layer to the original set of coordinates we should be able to assess if the matched.

Determine if nodes obtained from the OSM service were present on the OSRM Map

This planned activity consisted in using OSRM Nearest to check if the nodes in the database were present on the OSRM map.

We would pick random a node from the database, use its coordinates to make a request to the service, and check if the Node *osm_id* returned was the same. By repeating this operation numerous time, we would have a better assessment of the situation.

System Validation

We would test our system against the GraphHopper Traffic Data Integration Demonstration by making requests with similar coordinate inputs. The client script used in the last iteration experiences would be slightly modified to generate source and target coordinates inside the map, make a request to both services and store the results in a CSV file. By comparing these results we would be able to figure out whether or not our system was working properly.

Even though this would not validate our system, it would give us some indication of proper system operation.

E.5 Iteration 4

E.5.1 Risk Evolution

During the last iteration, we built an initial complete prototype and tested it against a similar system. This activity was met with mixed results. At this point, we would not make any further efforts to validate the system.

The other two activities resulted in promising results. The OSRM matching service allowed us to use the ORSM map as our project's single map source. This development was a major breakthrough for our project.

Even though this initial prototype was had proven the feasibility of using traffic sensor data to influence a routing engine, it was not robust enough to meet operational parameters. During this iteration, we would develop a more robust prototype.

E.5.2 Threshold of Success

The threshold of success remained the same.

E.5.3 Existing Risks Identification

Risk 2: Fulfilling our GIS data necessities using crowd funded map data

The GIS data necessities of the project were fulfilled by the highly normalized OSRM map. Working with a single map source also reduced the possibility of discrepancies. This feature improved our map data quality and reduced the probability of having further problems with map data.

Risk 2	Description
Source	Fulfilling our GIS data necessities with crowd sourced map data creates difficulties in map data usage
Condition	Data inaccuracy, missing tags in elements, lack of data validation
Consequence	May cause delays in system development and the system may be impossible to validate
Probability	Low
Impact	Critical
Time Frame	Short Term
Mitigation Plan	Remove irrelevant data and conform it to some hard standard

Table E.22: Risk 2 Evolution Iteration 4

Risk 7: Using nodes obtained from OSM Web Map API with the OSRM Experimental traffic feature

We adopted the OSRM map as our single map source. This risk no longer existed.

Risk 8: The system requires validation

It was decided that validating the system by testing it in real world conditions would be out of the scope of the project.

E.5.4 Risk Prioritization

	Likelihood	Low	Medium	High
Impact				
Catastrophic		Risk 5		
Critical		Risk 2, 3	Risk 1, 4	
Marginal				

Table E.23: Risk Exposure Matrix Iteration 4

Exposition Risk	Low	Medium	High	Critical

Table E.24: Exposition to Risk Iteration 4

ID	Exposition Risk	Time Frame
1	Medium	Short Term
4	Medium	Short Term
5	Medium	Medium Term
2	Low	Short Term
3	Low	Short Term

Table E.25: Risk Prioritization Iteration 4

E.5.5 Risk Mitigation Plan

At this point, we decided to develop a more robust system. We would start by making the necessary changes to the architecture to meet quality requirements. Then we would develop it according to this architecture. Finally, since this was our final iteration, we would thoroughly test our system.

We would start by performing unit testing the application server components, then we would evaluate if they integrated well with each other and with the rest of the components from the system. After deploying our system in a production environment, we would assess if the system was working properly and do usability tests to our browser-based frontend. Finally, we would assess if our prototype met all requirements and restrictions and could be accepted by Ubiwhere.

Appendix F

Iteration Four Architecture

F.1 Iteration 4 Architecture Layer View

Figure F.1: Iteration 4 Architecture Layer View

F.2 Tasks

F.2.1 Distributed Tasks Queue

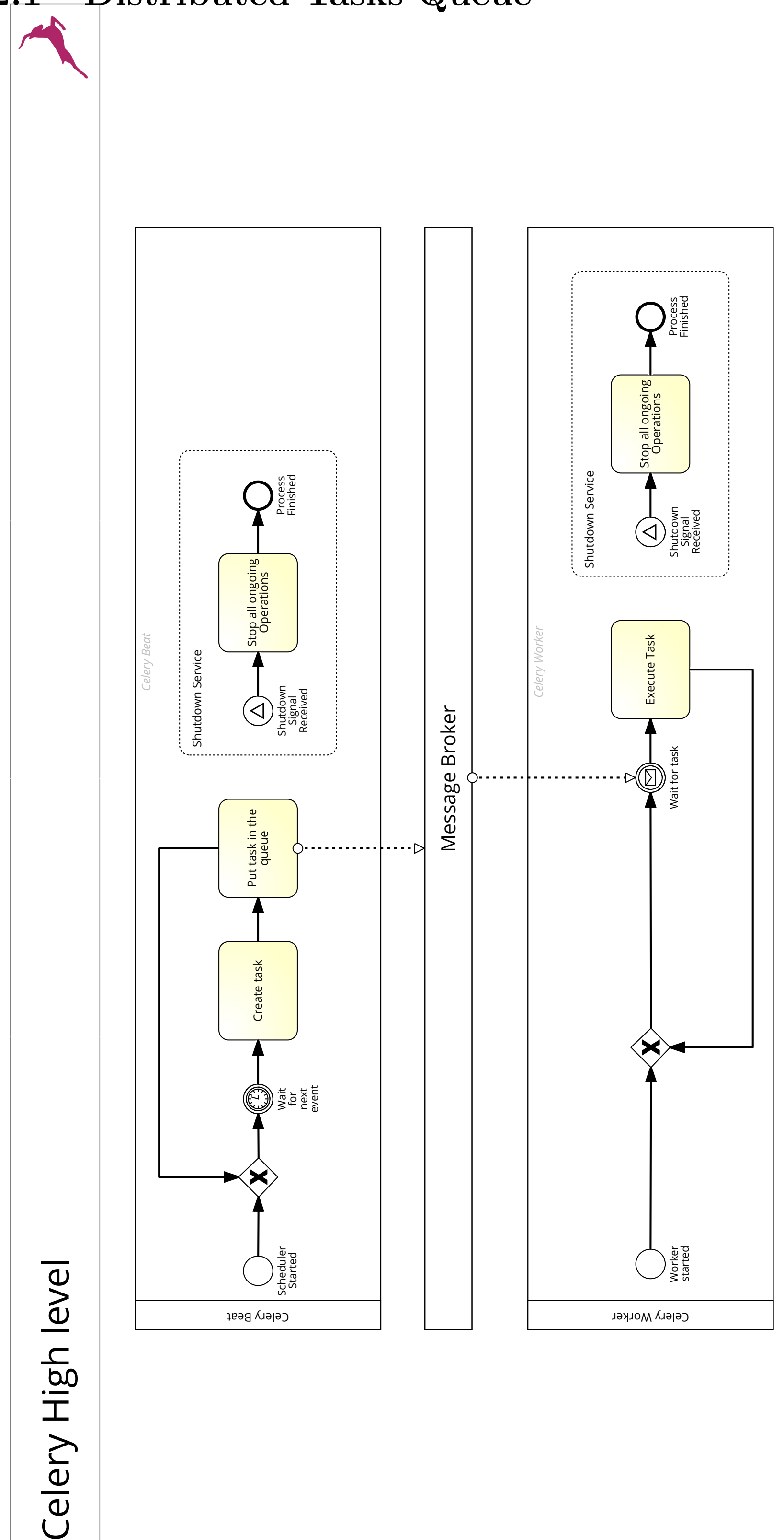


Figure F.2: Celery distributed tasks queue business process

F.2.2 Sensor Endpoint Task

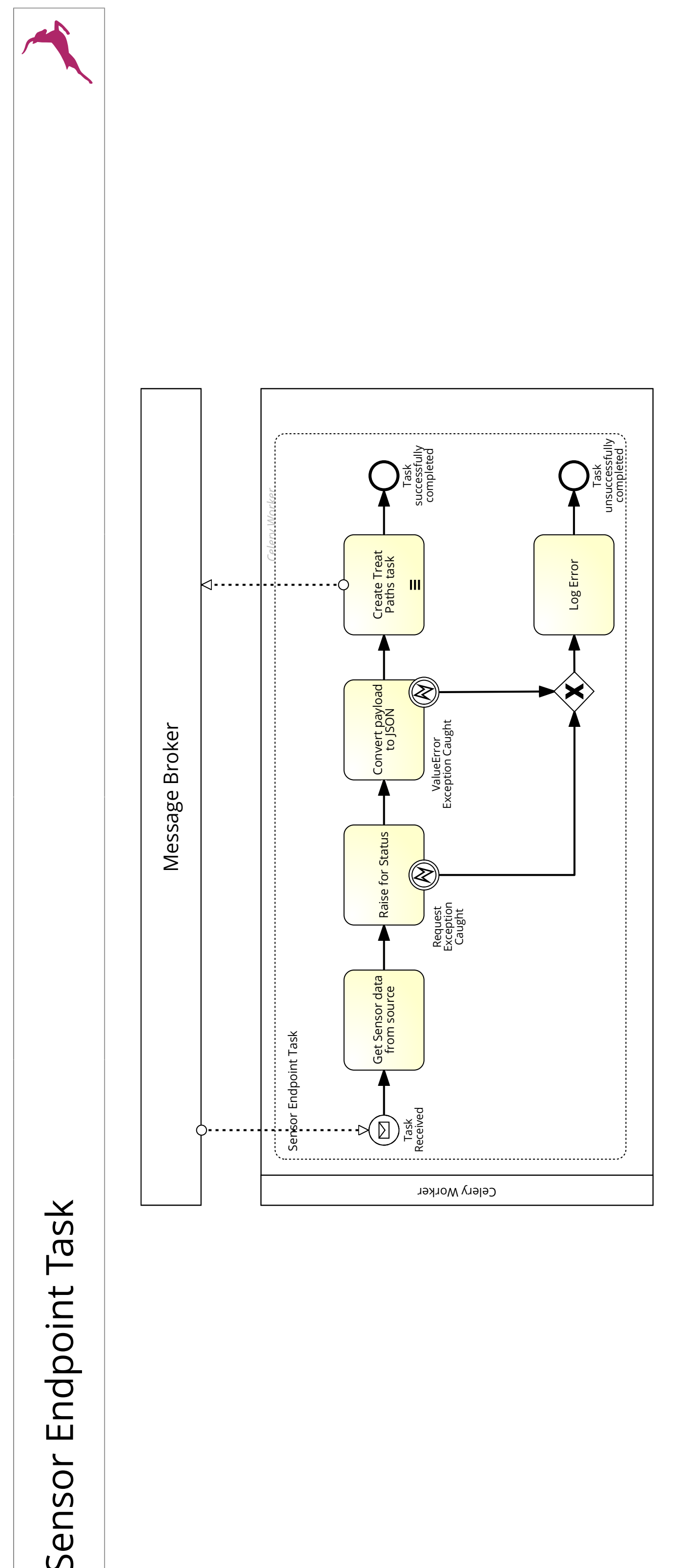


Figure F.3: Sensor Endpoint Task business process

F.2.3 Treat Paths Task

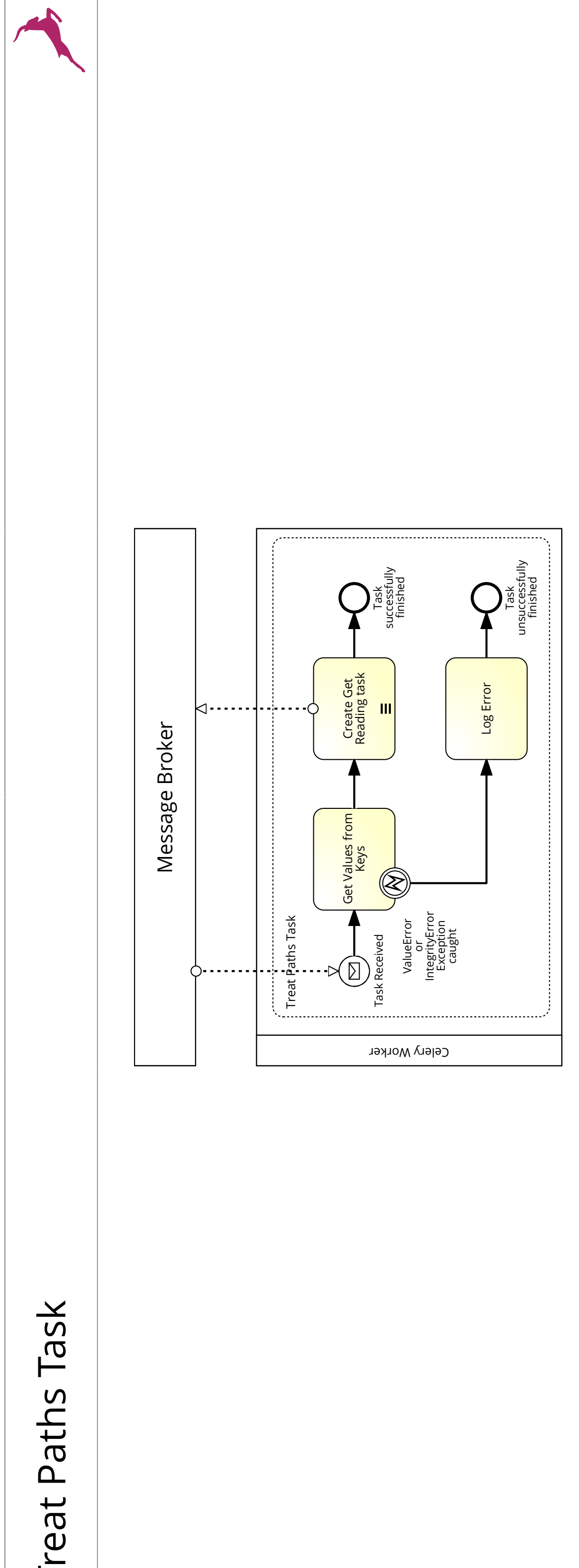


Figure F.4: Treat Paths Task business process

F.2.4 Get Reading Task

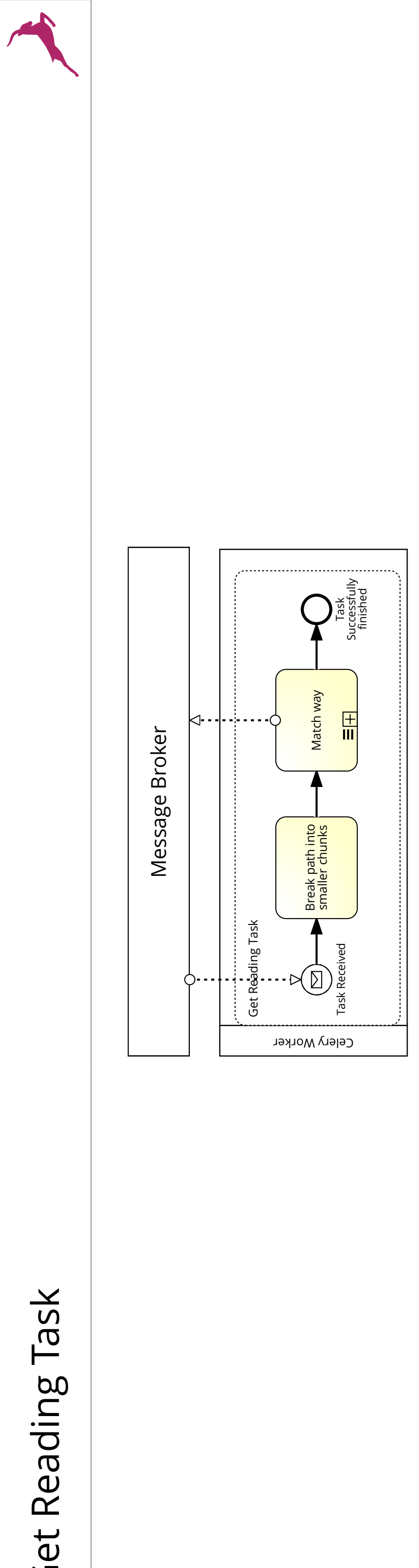


Figure F.5: Get Reading Task business process

F.2.5 Match Way Subprocess

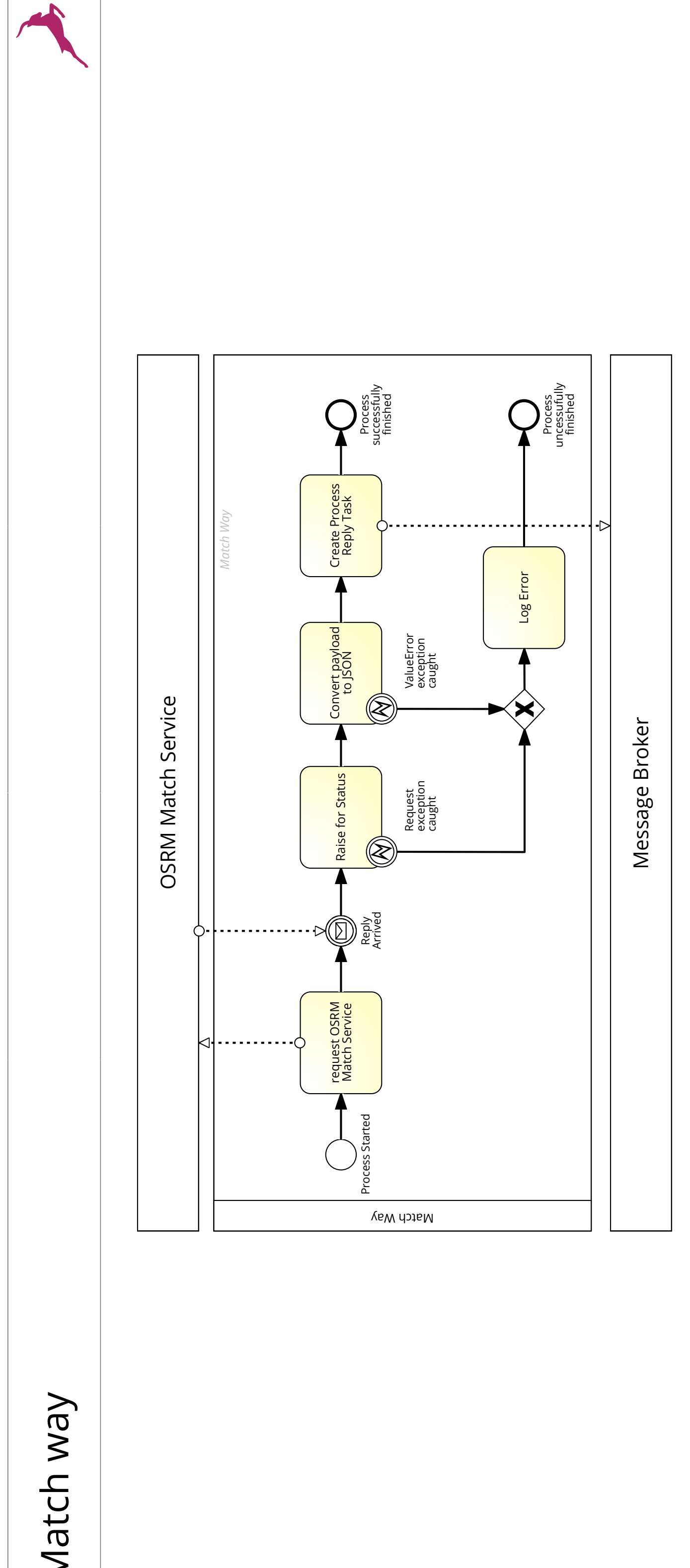


Figure F.6: Match way business sub process

F.2.6 Process Reply Task

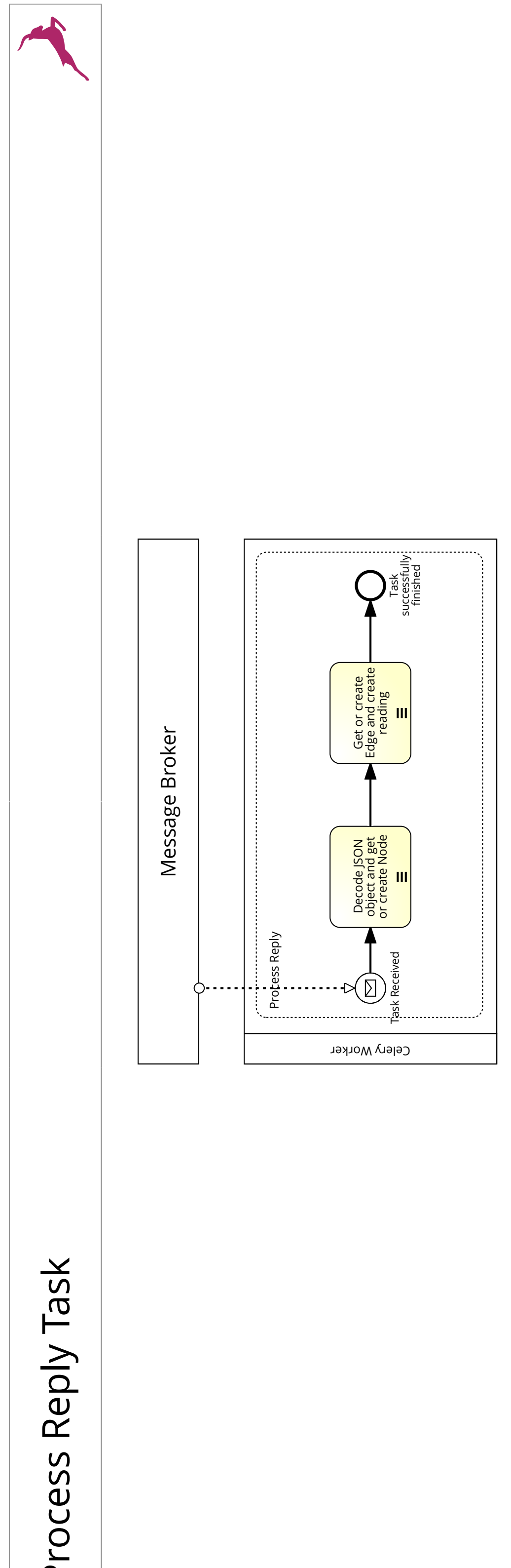


Figure F.7: Process Reply task business process

F.2.7 Compose Cache Objects Task

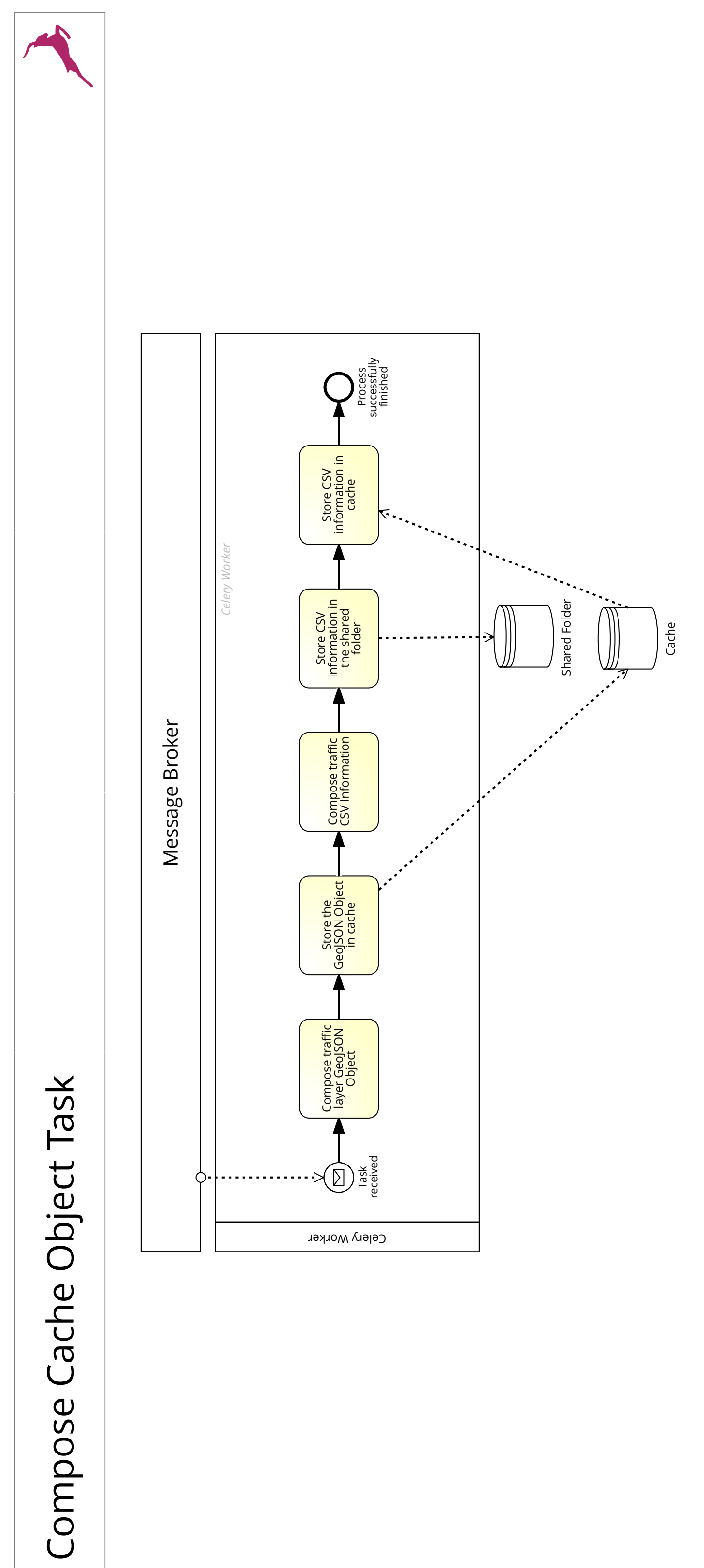


Figure F.8: Compose Cache Objects task business process

F.3 Routing Engine

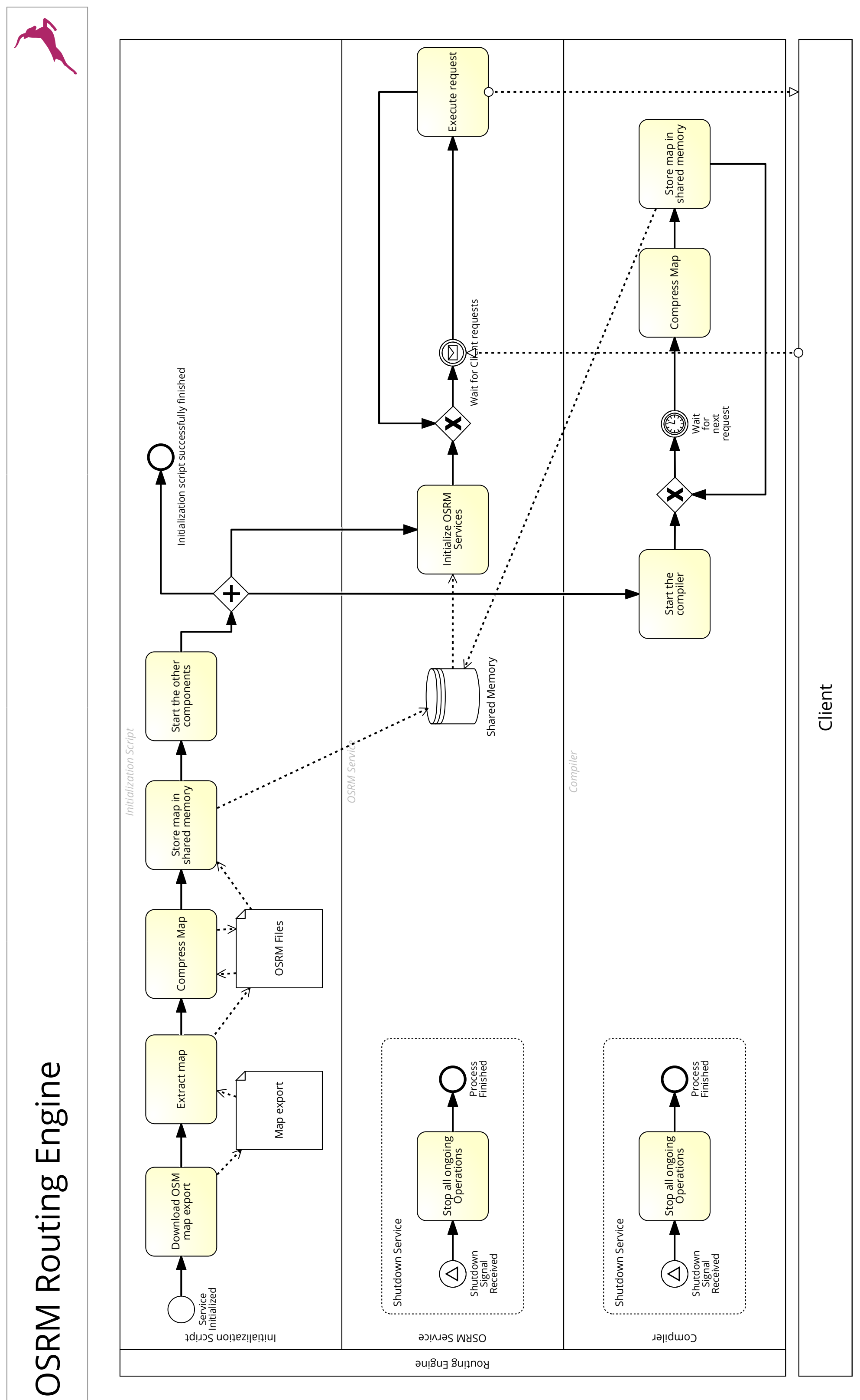


Figure F.9: Routing Engine business process