

# Masters Program in **Geospatial Technologies**



***Viability assessment of WPS 2.0 services as communication standard for expensive web-based machine learning analysis. A case of study: Indoor Location.***

Andrea Calia

Dissertation submitted in partial fulfilment of the requirements for the Degree of *Master of Science in Geospatial Technologies*

**Viability assessment of WPS 2.0  
services as communication standard for  
expensive web-based machine learning  
analysis. A case of study: Indoor  
Location.**

PhD Óscar Belmonte Fernández

*Institute of New Imaging Technologies*

*Universitat Jaume I,*

*Castellón de la Plana, Spain*

PhD PhD Marco Painho

*NOVA Information Management School*

*Universidade Nova de Lisboa,*

*Lisbon, Portugal*

PhD Raúl Montoliu Colás

*Institute of New Imaging Technologies*

*Universitat Jaume I,*

*Castellón de la Plana, Spain*

February 2016

## Acknowledgments

I would like to express my gratitude to PhD Óscar Belmonte Fernández, PhD Marco Painho and PhD Raúl Montoliu Colás for the dedication and help they provided to me for writing this thesis. Thanks to the teachers of the Master in Spain, Germany and Portugal.

I would like to extend my gratitude to the Geotec Research Group for giving me the opportunity to improve my skills and to meet great people. Special thanks to Nacho Miralles Tena. I also would like to thank my colleagues of the BE-OP team at CERN for the help and support they gave me in the last step of the Master.

Special thanks to my family who continuously gave me the support needed throughout my life. Finally, I would like to thank my girlfriend for her support and her patience.

# Abstract

Communication between client and server is a key factor in the modern age. Nowadays, telecommunications are at the base of every system and Software that is available. The way Software communicates can determine the efficacy of it.

In the GIS world, a server is often used for offloading expensive tasks such as geospatial operations or statistical analysis. This technique improves the performance of the Software systems and makes them able to scale based on the demand on real time.

For making the communication between client and server more efficient, interoperable and standard, the OGC released the standard WPS. WPS defines abstract operations that are able to describe a client server communication for remote process executions.

This thesis focuses on the asynchronous execution feature introduced in the version 2.0 of WPS. The main goal is to study how asynchronous process execution can benefit a client both in performance and availability.

The result are promising and it is demonstrated that WPS is a solid standard for describing web services operations. Based on the obtained results, future studies can extend the standard in order to make it more general and suitable for more situations.

# Keywords

async-wps

Classification

Geographical Information System

Indoor Location

Indoor Location Management Tool

Java

JavaScript

kNN

Machine Learning

Web Development

Web Processing Service

## Acronyms

API - Application Programming Interface	UI - User Interface
BDD - Behavior Driven Development	UML - Unified Modeling Language
CRS - Coordinate Reference System	URI - Universal Resource Identifier
CSS - Cascading Style Sheets	URL - Uniform Resource Locator
CSV - Comma Separated Values	VCS - Version Control System
GIS - Geographic Information System	WPS - Web Processing Service
GLONASS - GLObal NAVigation Satellite System	XML - Extensible Markup Language
GML - Geography Markup Language	
GNSS - Global Navigation Satellite System	
GPS - Global Positioning System	
GUI - Graphical User Interface	
HTML - HyperText Markup Language	
HTTP - Hypertext Transfer Protocol	
IDE - Integrated Development Environment	
IoT - Internet of Things	
ISO - International Organization for Standardization	
JSON - JavaScript Object Notation	
KVP - Keyword Value Pair	
MIME - Multipurpose Internet Mail Extensions	
OGC - Open Geospatial Consortium	
OSGEO - Open Source Geospatial Foundation	
OWS - OGC Web Services Common Standard	
SQL - Structured Query Language	

# Contents

<b>1</b>	<b>Introduction</b>	<b>14</b>
<b>2</b>	<b>State of the art</b>	<b>16</b>
<b>3</b>	<b>Web Processing Service 2.0</b>	<b>18</b>
3.1	Definition . . . . .	19
3.1.1	GetCapabilities . . . . .	20
3.1.2	DescribeProcess . . . . .	23
3.1.3	Execute . . . . .	26
3.1.4	GetStatus . . . . .	31
3.1.5	GetResult . . . . .	33
3.2	Interaction between Client and Server: Sync and Async execution models	36
3.2.1	Process discovery . . . . .	36
3.2.2	Process execution . . . . .	37
3.3	Issues and improvements . . . . .	40
<b>4</b>	<b>Web Processing Service Client Library: async-wps</b>	<b>41</b>
4.1	Current implementations . . . . .	42
4.1.1	52north-wps-js . . . . .	43
4.1.2	OpenLayers 2 . . . . .	43
4.2	Requirements . . . . .	45

4.3	Development and Results . . . . .	46
4.3.1	Technology and design decisions . . . . .	46
4.3.2	Requirements fulfillment . . . . .	48
4.3.3	Dependencies . . . . .	53
4.3.4	Tests . . . . .	54
4.3.5	Library compilation and test . . . . .	54
4.3.6	Source code . . . . .	55
4.4	Limitations . . . . .	56
<b>5</b>	<b>Case Study: Indoor Location</b>	<b>57</b>
5.1	Definition . . . . .	58
5.2	Classification problem . . . . .	59
5.3	WPS 2.0 server: wps-classicator . . . . .	60
5.3.1	Requirements . . . . .	60
5.3.2	Development and Results . . . . .	61
5.3.2.1	Technology and design decisions . . . . .	61
5.3.2.2	Requirement fulfillment . . . . .	62
5.3.2.3	Dependencies . . . . .	68
5.3.3	Source code . . . . .	69
5.4	Web Application: Indoor Location Management Tool . . . . .	70
5.4.1	Requirements . . . . .	70



5.4.2	Development and Results . . . . .	71
5.4.2.1	Technology and design decisions . . . . .	71
5.4.2.2	Requirement fulfillment . . . . .	73
5.4.2.3	Dependencies . . . . .	79
5.4.3	Limitations . . . . .	80
5.4.4	Source code . . . . .	81
<b>6</b>	<b>Discussion</b>	<b>82</b>
<b>7</b>	<b>Conclusion</b>	<b>87</b>
<b>8</b>	<b>Future work</b>	<b>90</b>

## List of Figures

1	Conceptual model of a WPS server. From [28] . . . . .	20
2	Process discovery interaction between a client and a WPS 2.0 compliant server. From Figure 15 of [28] . . . . .	37
3	Process execution interaction between a client and a WPS 2.0 compliant server. (a) shows the interaction using synchronous request, while (b) using an asynchronous request. From Figure 3 and 4 of [28] . . . . .	39
4	The tabular view visualizes the element of the database in rows. Each column represent a attribute of the element. Pagination is added because the expected number of entries of the database is large. The page size can be customized . . . . .	74
5	The map view visualize the data taking into account the geospatial attributes of the elements. Those are latitude and longitude specified using WGS84. Custom layers can be added into the map to better understand and analyze the data. Elements that are very close to each other in the selected zoom are clustered together for clarity reasons . . . . .	75
6	Filtering options using a form. It is possible to filter the data based on non-geospatial properties. In this early version, the filter options occurs comparing using equality the properties set in the form with the ones stored in the database . . . . .	76
7	Filtering options for geospatial attributes. The filter happens using the geospatial operation "within". The user chooses the preferred method (arbitrary polygon or rectangle), draws the feature on the map and the database entries that fall into the feature are selected automatically . . .	77
8	The experiment details are shown in the sidebar of the web application. In the central part, the elements of the experiment are visualized . . . . .	78

9	This chart compares the status of a WPS client during asynchronous (a) and synchronous (b) requests for the Large dataset. The simulation is particularly heavy since it involves 15000 elements . . . . .	84
10	This chart compares the status of a WPS client during asynchronous (a) and synchronous (b) requests for the Small dataset. The simulation is particularly heavy since it involves 15000 elements . . . . .	85
11	These charts show the evolution of the completing time for the proposed simulations compared with the interval between status polling. (a) shows the results for the Large dataset, while (b) shows the results using the Small dataset. The polling interval (in seconds) is the time the client waits before requesting the status of the execution. The selected polling interval affects significantly the total execution time of the experiments depending on the duration of the experiment (in seconds). Longer experiments (execution time) seems not to be heavily affected . . . . .	86

## List of Tables

1	Specific properties of a GetCapabilities request. Based on Table 35 of WPS 2.0 standard specification [28] . . . . .	21
2	Specific properties of a GetCapabilities response. Based on Table 36 of WPS 2.0 standard specification [28] . . . . .	21
4	Specification of a Process Summary property. Based on Table 37 and 29 of WPS 2.0 standard specification [28] . . . . .	22
5	Specific properties of a DescribeProcess request. Based on Table 38 of WPS 2.0 standard specification [28] . . . . .	23
6	Specific properties of a DescribeProcess response. Based on Table 39 of WPS 2.0 standard specification [28] . . . . .	24
8	Specification of a Process Offerings property. Based on Table 40 and 29 of WPS 2.0 standard specification [28] . . . . .	25
9	Specific exceptions for a DescribeProcess request. Based on Table 41 of WPS 2.0 standard specification [28] . . . . .	25
10	Specific properties of an Execute request. Based on Table 42 of WPS 2.0 standard specification [28] . . . . .	27
13	Specification of a Data Output Definition Type. Based on Table 44 and 24 of WPS 2.0 standard specification [28] . . . . .	28
14	Specific exceptions for a Execution request. Based on Table 46 of WPS 2.0 standard specification [28] . . . . .	30
12	Specification of a Data Input Type. Based on Table 43 of WPS 2.0 standard specification [28] . . . . .	31

15	Specific properties of a GetStatus request. Based on Table 47 of WPS 2.0 standard specification [28] . . . . .	32
16	Specification of a Status Info document. Based on Table 32 of WPS 2.0 standard specification [28] . . . . .	33
17	Specific exceptions of a GetStatus request. Based on Table 48 of WPS 2.0 standard specification [28] . . . . .	33
18	Specific properties of a GetResult request. Based on Table 49 of WPS 2.0 standard specification [28] . . . . .	34
19	Specific properties of a GetResult response. Based on Table 33 of WPS 2.0 standard specification [28] . . . . .	34
20	Specification of a Data Output Type. Based on Table 34 of WPS 2.0 standard specification [28] . . . . .	35
21	Specific exceptions of a GetResult request. Based on Table 50 and 48 of WPS 2.0 standard specification [28] . . . . .	35
22	Summary of the requirements for the async-wps library . . . . .	45
23	Summary of the requirements for the wps-classificator . . . . .	60
24	Summary of the requirements for the Indoor Location Management Tool	71
25	This table shows the status of the client during an asynchronous and a synchronous request for each simulation. The large dataset involves 15000 elements, while the Small dataset only 3000 elements. During the Blocked status, the client cannot perform any action, since it is waiting for the response of the server. During the Idle status, the client is available for doing other tasks, like updating the GUI or processing other data. The time is expressed in seconds . . . . .	83

# 1 Introduction

The amount and heterogeneity of geospatial data has increased drastically in the past few years. This is due to the improvements on network performance and the capabilities of modern hardware [21]. At the beginning of the Internet era, the digital representation of geospatial data was very simple, but nowadays it become a complex task to digitalize and maintain geospatial data. The main motivation is that geospatial information is now growing exponentially due to its value in, for example, business, climate change analysis and socio-economical development. This situation created the need of developing new data formats, analysis algorithms and delivery standards. This heterogeneity leads to complex Software architectures and confusions for the final user [24].

There were multiple attempts to make the development of geospatial Software applications less complex. One of the most recognised is the release of the Web Processing Service (WPS) [26] by the Open Geospatial Consortium in 2007. This standard introduces a way to publish, discover and execute geospatial algorithm through the Web[14]. The power of the standard is the interoperability it introduces for executing processes on remote servers. Being the discovery, communication and format in a standardized way, any client can interact with the server, exchange information or schedule processes execution.

According to the standard WPS 1.0, the execution of a process happens synchronously. This means that the client send a request for a certain process to execute and wait for the server response. This mechanism is very simple but it has problems when the process needs or produces large quantity of data or it is particularly time consuming. In such a situation, the client is blocked for a long time waiting for the server response. This issue can cause some problems like the faulty user responsiveness, network timeout or computational resource waste.

To improve this situation, OGC released in 2015 the version 2.0 of the WPS standard [28]. This new revision addresses the issues raised by the community over the first version of the standard. One of the main improvement is the support for asynchronous operations

as part of the standard. A client is now able to schedule a process execution on a server and continue its execution. Periodically, the client will query the server for the execution status and, eventually, for the result.

The hypothesis of this thesis is to take advantage of the asynchronous execution capabilities of WPS 2.0 for improving the performance and the user experience of web geospatial applications. This is of particular interest in cases where long running algorithms are performed. This can be the case of expensive geospatial operations, machine learning algorithm, clustering and statistical analysis with large datasets.

In order to achieve this goal, a client implementation of the WPS standard is presented. This library is called `async-wps` and is able to communicate with WPS 2.0 compliant servers in a asynchronous way. Also, it exposes a simple API for query and execute processes on the server (see Section 4.2 for the complete list of the requirements).

In the direction of demonstrating the proposed hypothesis, the case of study of Indoor Location is proposed. The case of study includes a WPS 2.0 server (`wps-classificator`) that implements the kNN classification algorithm and a web application that manage a Indoor Location database (Indoor Location Management Tool) [35].

The following sections are structured as follow: Section 2 provides a description of the usage of WPS standard in the academic literature. Section 3 provides a summary of the WPS 2.0 standard meant as an overview of the specification to better understand the context of the thesis. Section 4 describe in details the Software library `async-wps`. Section 5 provides a description of the case of study, including the server `wps-classificator` and the Indoor Location Management Tool. Section 6 shows the discussion of the results and the improvement of the client responsiveness and performance. Section 7 explain the conclusions of the proposed thesis. Finally, Section 8 shows the following step to be undertaken in the future as the result of the work of this thesis.

## 2 State of the art

WPS is a well recognised standard and is been used in a variety of projects successfully. Despite this, WPS defines abstract operations and data formats, but does not provide official implementations ready to be used [10].

In [13], the problem of connecting multiple WPS endpoints is undertaken. The proposed approach involves the extension of the WPS standard for including two parameters. The proposed implementation is a proof of concept and is able to mediate different WPS services in a computing Grid. The context of the work is the calculation of the Normalized Difference Vegetation Index.

The work proposed in [5] uses WPS standard for providing high performance and scalable web service for analysing Earth Observation data. In such a context, WPS is able to orchestrate the communication between client and servers in order to provide more throughput in the data analysis. Also, a multilayer framework of WPS servers is able to manage the user requests and the cloud computing environment. To be able to process large quantity of data in real time, the cloud computing layer is built on top of Apache Hadoop.

An interesting project is found in the work of Fenoy et all (20013) in which the ZOO Project is developed [10]. In this project, a WPS framework is build with the aim to integrate and link various algorithms and expose them using WPS interface. It provides connectors for developing algorithms using C/C++, Java, Perl, Python, PHP and JavaScript. This framework supports only the version 1.0 of WPS.

In [2], the WPS standard is used for creating a service oriented infrastructure. The proposed problem involves the usage and coupling of various mathematical models in the context of Environmental modelling. In this scenario, each mathematical model is represented by a WPS web service. Then, a client is able to orchestrate the workflow because every web service use a standard WPS interface. The proposed system is based on the version 1.0 of WPS.



Stollberg et al. [33] presents the aggregation of WPS web services in a Service oriented Architecture. In this architecture, the WPS services are able to communicate between each other for exchanging information. For the first time, the term "Composite-WPS" is introduced. This term indicates the chaining of dependency of a specific WPS process. This allows, for example, to chain multiple WPS web services and, when invoked, they will be triggered automatically. This architecture is validated in a simulated bomb treatment scenario in the context of quick response to a disaster.

Degree Framework [12] is a project that allows the easy development of WPS servers using Java as programming language. It has the particularity of supporting large amount of data using streaming techniques. This framework supports most of OGC standards (e.g. GML, WFS, WFS, WMS and WPS) in order to provide a full-stack open source software for spatial data infrastructure. The component offered by the framework include the management, visualization and access to geospatial information. The supported version of WPS is 1.0.

Another open source WPS framework for Java programming language is WPSint [37]. It provides an easy integration with Spring<sup>1</sup>. This framework is meant to be used to easily develop web services and expose them in a way compatible with WPS. The project, even if it has some interesting features, is no longer maintained and it supports only the WPS standard version 0.4.0.

For accessing WPS servers using JavaScript, there are two libraries: 52° North WPS [38] and OpenLayers2 [15]. The first one is JavaScript project that offers a web interface for accessing WPS servers. Its usage is limited because the WPS client code is tightly coupled with the project and is not proposed as external module. The supported version of WPS is 1.0. The latter is a famous JavaScript library for creating geospatial web applications. It aims to be full-features including an implementation for the majority of OGC standards. WPS compatibility is only present in the second version of the library and the it is limited to WPS 1.0. Section 4 explain them in details

---

<sup>1</sup><https://spring.io/>

### 3 Web Processing Service 2.0

This section covers the WPS 2.0 standard. Its aim is to provide an overview of the core operations and the basic interactions between a client and a server that use WPS as a communication protocol. Its aim is to provide a simplified version to better understand the standard. The properties included refers only to the ones specified in the WPS 2.0 standard specification, to access or create a fully functional WPS 2.0 server it may involve to include options and properties of others OGC standards, such as OWS Commons [27].

### 3.1 Definition

WPS is a standard developed at the OGC.

According to its website[29], OGC defines itself as "*an international industry consortium of over 520 companies, government agencies and universities participating in a consensus process to develop publicly available interface standards*". The main area of concern of OGC is geospatial information. One of the goals is to provide and improve, where exists, interoperability between geospatial software and services.

WPS defines an abstract framework of communication between two entities, the client and the server. In this scenario, the server offers a particular process that the client needs to execute. As WPS is meant to be used on the Web the client and the server are normally distributed and may not reside on the same machine. For example, the server can belong to a private company that sells services or to a public university that provides services free of charge.

The latest version of WPS is 2.0[28] and it has been published on the fifth of march of 2015. This revision of the standard introduces some significant advantages, such as: flexibility to be used using REST or SOAP; nested inputs and nested outputs; better process description; synchronous and asynchronous execution. To be able to support the new features, data and process models have been revised.

Figure 1 shows the conceptual model of a WPS server. It can be summarized into:

- Web Processing Service: the server itself.
- Process: the operation that the server provides. For example, a Buffer or a Convex Hull operation.
- Job: it represent and instance of an operation that executes a specified process with associated data.
- Job Control Operation: it provides the ability to control a job that is executing

on the server. It can provide such operations as: load balancing; error report; log; dismiss; etc.

- Data: data that the job uses to operate.
- Status: the current status of the job.

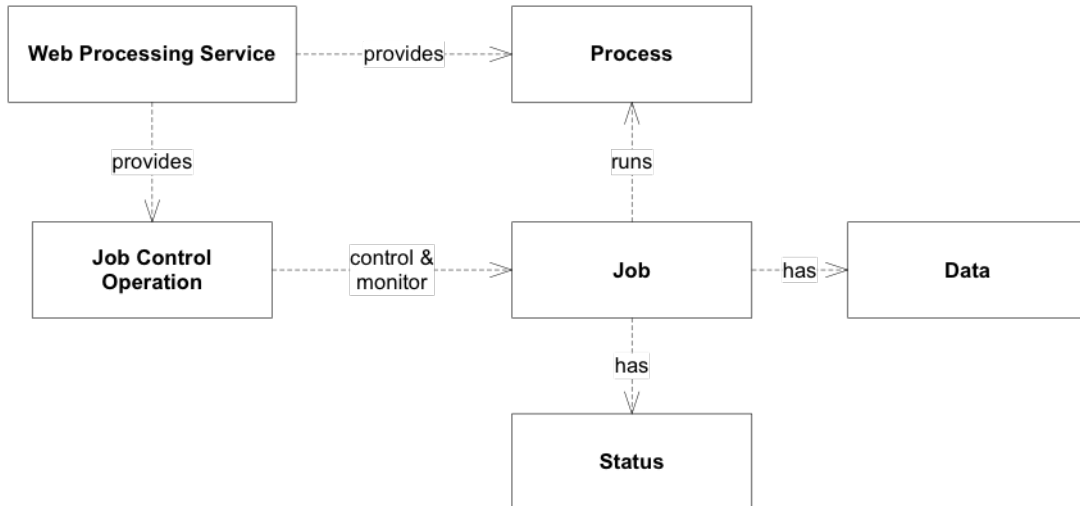


Figure 1: Conceptual model of a WPS server. From [28]

Version 2.0 of WPS standard defines five operations that a compliant server needs to implement: GetCapabilities; DescribeProcess; Execute; GetStatus; GetResult.

### 3.1.1 GetCapabilities

GetCapabilities operation allows the client to get metadata and information about the processes available on the WPS server. This operation is particularly useful in two scenarios: capabilities discovering of an unknown WPS server; checking for availability of a specific process of a known WPS server.

Table 1 shows the properties of a WPS GetCapabilities request. Table 2 shows the properties of a WPS GetCapabilities response.

In the case of an error, the WPS server should respond with a generic exception (specification at [27]).

<b>Name</b>	<b>Description</b>	<b>Type</b>	<b>Cardinality</b>	<b>Mandatory</b>
Service	Service identifier	Fixed character string: "WPS"	One	Yes
Extension	Hook for future extension specifications	Any	Zero or more	No

Table 1: Specific properties of a GetCapabilities request. Based on Table 35 of WPS 2.0 standard specification [28]

<b>Name</b>	<b>Description</b>	<b>Type</b>	<b>Cardinality</b>	<b>Mandatory</b>
Service	Service identifier	Fixed character string: "WPS"	One	Yes
Contents	Contains a list of the offered processes.	ProcessSummary, Table 4	One	Yes
Extension	Hook for future extension specifications	Any	Zero or more	No

Table 2: Specific properties of a GetCapabilities response. Based on Table 36 of WPS 2.0 standard specification [28]

<b>Name</b>	<b>Description</b>	<b>Type</b>	<b>Cardinality</b>	<b>Mandatory</b>
Title	Name of the process	String	One	Yes
Abstract	Brief description of the process	String	Zero or more	No
Keywords	List of keywords that describe better the process	String	Zero or more	No
Identifier	Unambiguous identifier for the process	String	One	Yes
Job Control Options	Options for controlling this process	List of options: "sync-execute"; "async-execute"; "dismiss"	One or more	Yes
Output Transmission	Transmission modes for output data	List of transmission options: "value"; "reference"	One or more	Yes
Process Model	Type of the process description	URI	Zero or one	No

Table 4: Specification of a Process Summary property. Based on Table 37 and 29 of WPS 2.0 standard specification [28]

### 3.1.2 DescribeProcess

DescribeProcess operation allows a client to get a detailed description of a process on a WPS compliant server. This information includes the description of the inputs and outputs of the process. It is very important to check this information before executing an operation, otherwise it may lead to unexpected behavior.

Table 5 shows the properties of a DescribeProcess request. Table 6 shows the properties of a DescribeProcess response.

In the case of an error, the WPS server should respond with an exception. Table 9 shows the possible exceptions for a DescribeProcess request.

Name	Description	Type	Cardinality	Mandatory
Identifier	Process identifier	String (special value "ALL" indicates that the will to retrieve the description for all the processes)	One or more	Yes
Lang	Indicates the language for the desired description	String in the human-readable format from the standard IETF RFC 4646 [16]	Zero or one	No

Table 5: Specific properties of a DescribeProcess request. Based on Table 38 of WPS 2.0 standard specification [28]

<b>Name</b>	<b>Description</b>	<b>Type</b>	<b>Cardinality</b>	<b>Mandatory</b>
Lang	Indicates the language for the desired description	String in the human-readable format from the standard IETF RFC 4646 [16]	Zero or one	No
Process Offerings	Contains a list of the description for each queried process	ProcessOffering, Table 8	One or more	No

Table 6: Specific properties of a DescribeProcess response. Based on Table 39 of WPS 2.0 standard specification [28]



<b>Name</b>	<b>Description</b>	<b>Type</b>	<b>Cardinality</b>	<b>Mandatory</b>
Any	Any property it may be useful to better describe the process.	Any	One or more	Yes
Job Control Options	Options for controlling this process	List of options: "sync-execute"; "async-execute"; "dismiss"	One or more	Yes
Output Transmission	Transmission modes for output data	List of transmission options: "value"; "reference"	One or more	Yes
Process Model	Type of the process description	URI	Zero or one	No

Table 8: Specification of a Process Offerings property. Based on Table 40 and 29 of WPS 2.0 standard specification [28]

<b>Code</b>	<b>Description</b>	<b>HTTP Code</b>
NoSuchProcess	The WPS server does not contain one or more queried processes	400 (Bad request)

Table 9: Specific exceptions for a DescribeProcess request. Based on Table 41 of WPS 2.0 standard specification [28]

### 3.1.3 Execute

Execute operation allows a client to execute an operation provided by a WPS compliant server. This operation schedules a job on the server that runs the operation with the given inputs (if any) and, eventually, returns an output.

The result may be returned either by value or by reference. If the output is returned by value, it is embedded in the response of the Execute request or GetResult (see Section 3.1.5) request. A reference return type means that the output is stored in an web accessible way. For example, in a different server or in a different domain. It must be specified by a URI and it must allow the download using standard HTTP. The latter case is very useful in the situation that involves a large amount of data as a output. In this case, the client may want to schedule the download of the result in a different moment. Also, it can improve the overall performance of the server by serving the result over a CDN or a dedicated Web Service that does not slow down the jobs execution time.

A major feature of the version 2.0 of WPS standard is the ability to request synchronous (Sync) and asynchronous (Async) execution models for a process. In a synchronous execution model, an Execution request from a client is blocked until the process is done and, either a result is produced or an error is thrown. In contrast, the asynchronous execution model allows the client to schedule a job on the server and then to continue its calculations/operations. At some point in the future, the client will query the server for the status and the result of the scheduled job. Section 3.2 explains in details the interaction between a WPS client and server in both Sync and Async execution models.

Table 5 shows the properties of an Execute request. The response of an Execute operation is based on the server implementation and the type of requested operation. For an Async operation, the response is a Status Info document (Table 16). This document describes the current status of a scheduled job on the server. This document is explained in details in Section 3.1.4. In the case of a Sync operation, the result may be the response document with the result of the operation or raw data. The latter case is allowed only

when the process returns a single output and there are no errors.

In the case of an error, the WPS server should respond with an exception. Table 14 shows the possible exceptions for an Execute request.

<b>Name</b>	<b>Description</b>	<b>Type</b>	<b>Cardinality</b>	<b>Mandatory</b>
Identifier	Process identifier	String	One	Yes
Mode	Indicates the preferred execution model for the process	Fixed string, one of: "sync"; "async"; "auto"	One	No
Response	Indicates the preferred response format	Fixed string, one of: "document"; "raw". The latter can be specified only when the process returns a single output	One	No
Input	Contains the list of the inputs for the process	Data Input Type, Table 12	Zero or one	No
Output	Contains the list that specify the desired format of each output	Output Definition Type, Table 13	Zero or one	No

Table 10: Specific properties of an Execute request. Based on Table 42 of WPS 2.0 standard specification [28]

<b>Name</b>	<b>Description</b>	<b>Type</b>	<b>Cardinality</b>	<b>Mandatory</b>
Id	Identifier for the output	URI	One	Yes

Transmission	Desired transmission mode. The valid transmission modes are listed on the Output Transmission property of a Describe Process query (Table 6 and Table 8)	String	Zero or one	No
Mimetype	Desired Mimetype for this output	String	One	Yes
Encoding	Desired encoding for this output	String	Zero or one	No
Schema	Data schema for the output	URI	Zero or one	No
Output	A nested output element	Data Output Type, this table	Zero or one	No

Table 13: Specification of a Data Output Definition Type. Based on Table 44 and 24 of WPS 2.0 standard specification [28]

Code	Description	HTTP Code
NoSuchProcess	The WPS server does not contain one or more queried processes	400 (Bad request)
NoSuchMode	The process does not allow the specified execution mode	400 (Bad request)

NoSuchInput	One or more input specified on the request does not match with the process description	400 (Bad request)
NoSuchOutput	One or more output specified on the request does not match with the process description	400 (Bad request)
DataNotAccessible	One or more input is not accessible. This applies in the case of input specified as Reference	400 (Bad request)
SizeExceeded	One or more input cannot be handled due to its size	400 (Bad request)
TooManyInputs	There are more inputs than the ones specified on the process description	400 (Bad request)
TooManyOutputs	There are more outputs than the ones specified on the process description	400 (Bad request)
ServerBusy	The server cannot accept any more requests	503 (Service Unavailable)
StorageNotSupported	The server does not support transmission mode Reference	400 (Bad request)
NoSuchFormat	One or more inputs or outputs did not match the format described on the process description	400 (Bad request)
WrongInputData	The received input data cannot be read	400 (Bad request)
InternalServerError	Internal server error	500 (Internal Server Error)

---

Table 14: Specific exceptions for a Execution request. Based on Table 46 of WPS 2.0 standard specification [28]

Name	Description	Type	Cardinality	Mandatory
Id	Identifier for the input	URI	One	Yes
Data	Embedded data	WPS Data structure, Table 23 of [28]	Zero or one	Yes, only one should be used
Reference	Reference of where the data should be fetched from	WPS Reference structure, Table 25 of [28]	Zero or one	
Input	A nested input element	Data Input Type, this table	Zero or one	

Table 12: Specification of a Data Input Type. Based on Table 43 of WPS 2.0 standard specification [28]

### 3.1.4 GetStatus

The GetStatus operation allows a client to retrieve the status of an asynchronous job that is running on a server. This operation should be done after scheduling an execution using the Execute operation, specifying the Async execution mode.

Table 15 shows the properties of a WPS GetStatus request. The result of a GetStatus request is a StatusInfo document, described in Table 16. The StatusInfo document is used to determine which is the current state of the scheduled job on the server. WPS 2.0 standard defines four types of state (Table 3 of WPS 2.0 standard specification[28]):

- Succeeded: The job is done and no error occurred.
- Accepted: The job has been accepted by the server and it is scheduled for execution.
- Running: The job is currently running on the server.
- Failed: The job produced an error during its execution.

A WPS 2.0 server can provide custom states for a job. The standard does not limit the number of possible states and allows domain-specific states in the case they are meaningful. For example, a server with very high demand for long-running tasks, may include the state "Paused". This information can be used by the client to provide feedback to the end user of an application improving its user-friendliness.

Table 17 shows the exceptions a WPS 2.0 compliant server should throw in the case of an error during a GetStatus request.

<b>Name</b>	<b>Description</b>	<b>Type</b>	<b>Cardinality</b>	<b>Mandatory</b>
Job ID	The identifier of a job scheduled on the server	String	One	Yes

Table 15: Specific properties of a GetStatus request. Based on Table 47 of WPS 2.0 standard specification [28]

<b>Name</b>	<b>Description</b>	<b>Type</b>	<b>Cardinality</b>	<b>Mandatory</b>
Job ID	Job identifiers	String	One	Yes
Status	Indicates the current status of the specified job	String, it may include one of the following: "Succeeded"; "Failed"; "Accepted"; "Running"	One	Yes
Expiration Date	It specifies when the job and its results will no longer be accessible	String following ISO-8601 [1] standard	Zero or one	No



Estimated Completion	Com-	It specifies when the job is expected to be done	String following ISO-8601 [1] standard	Zero or one	No
Next Poll		It specifies when it is recommended to request a status update	String following ISO-8601 [1] standard	Zero or one	No
Percent Completed	Com-	The percentage of completion of this job	Integer from 0 to 1	Zero or one	No

Table 16: Specification of a Status Info document. Based on Table 32 of WPS 2.0 standard specification [28]

Code	Description	HTTP Code
NoSuchJob	The WPS server does not have any job with the specified identifier	400 (Bad request)

Table 17: Specific exceptions of a GetStatus request. Based on Table 48 of WPS 2.0 standard specification [28]

### 3.1.5 GetResult

The GetResult operation it is intended to be used to retrieve the result of an asynchronous job execution on a WPS 2.0 server.

Table 18 shows the properties of a GetResult request. Table 19 shows the properties of a GetResult response.

In an error occurred during a GetResult operation, the server should return an exception. Table 21 shows the specific exceptions for a GetResult operation.

<b>Name</b>	<b>Description</b>	<b>Type</b>	<b>Cardinality</b>	<b>Mandatory</b>
Job ID	The identifier of a job scheduled on the server	String	One	Yes

Table 18: Specific properties of a GetResult request. Based on Table 49 of WPS 2.0 standard specification [28]

<b>Name</b>	<b>Description</b>	<b>Type</b>	<b>Cardinality</b>	<b>Mandatory</b>
Job ID	The identifier of a job scheduled on the server	String	One	Yes
Expiration Date	It specifies when the job and its results will no longer be accessible	String following ISO-8601 [1] standard	Zero or one	No
Output	Contains a list with the outputs for the specified job	Data Output Type, Table 20	One or more	Yes

Table 19: Specific properties of a GetResult response. Based on Table 33 of WPS 2.0 standard specification [28]

<b>Name</b>	<b>Description</b>	<b>Type</b>	<b>Cardinality</b>	<b>Mandatory</b>
ID	Identifier of the output	String	One	Yes
Data	Embedded data	Any	Zero or one	Yes, only one should be used
Output	A nested output element	Data Output Type, this table	Zero or one	

Table 20: Specification of a Data Output Type. Based on Table 34 of WPS 2.0 standard specification [28]

<b>Code</b>	<b>Description</b>	<b>HTTP Code</b>
NoSuchJob	The WPS server does not have any job with the specified identifier	400 (Bad request)
ResultNotReady	The specified job has not yet finished its execution	400 (Bad request)

Table 21: Specific exceptions of a GetResult request. Based on Table 50 and 48 of WPS 2.0 standard specification [28]

## 3.2 Interaction between Client and Server: Sync and Async execution models

The WPS standard is an abstract communication protocol between a client and a server. The aim of the client is to execute an operation (service) offered by the server. The WPS standard provides abstract operation for discovery, description and execution of a operation (service).

As stated in Section 3.1, WPS 2.0 defines five operations: GetCapabilities; DescribeProcess; Execute; GetStatus; GetResult. The interaction between a client and a server in the context of WPS can be divided into two sections: process discovery and process execution.

### 3.2.1 Process discovery

The WPS 2.0 operations involved in the process discovery are: GetCapabilities and DescribeProcess. A client needs to use those operations in order to know the processes offered by a WPS 2.0 server.

The first step for a WPS client is to use the GetCapabilities operation. The result of this operation includes the metadata about the server, including the processes it offers. For example, in addition to the offered processes, a WPS server may specify an abstract that describes the included services, some keywords, the supported version of WPS and a contact person in case of error or misbehavior.

With the information of the GetCapabilities operation, the client may select a suitable process that solves its needs. If it is the case, a DescribeProcess operation should be issued. This operation retrieves the metadata for the specific process of the server. In this way, the client knows exactly which input it should provide and which output it should expect. Another important information is the available execution models, for example it may be convenient to check if the server supports asynchronous execution model.

Figure 2 shows a UML interaction diagram that summarize the process discovery procedure.

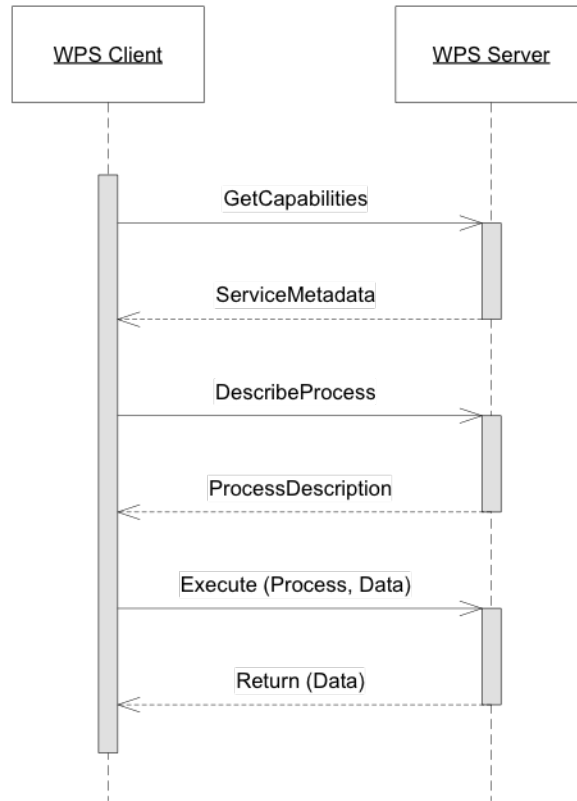


Figure 2: Process discovery interaction between a client and a WPS 2.0 compliant server. From Figure 15 of [28]

### 3.2.2 Process execution

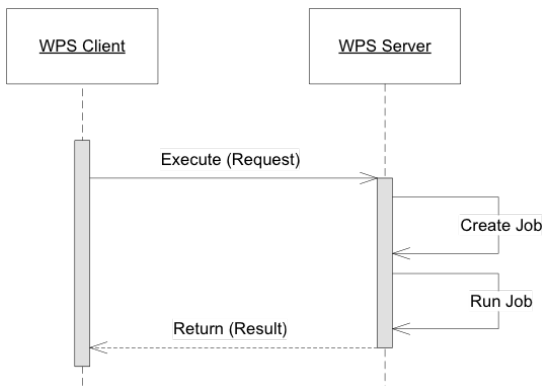
The process execution phase include the execution of a previously selected process available on the WPS server. Therefore, it is assumed that the client knows the identifier of the job it needs to access (if this is not the case, see Section 3.2.1).

The version 2.0 of the WPS standard [28] introduces, among other features, the ability to specify the execution model of a process. There are two execution model defined in the standard: synchronous and asynchronous.

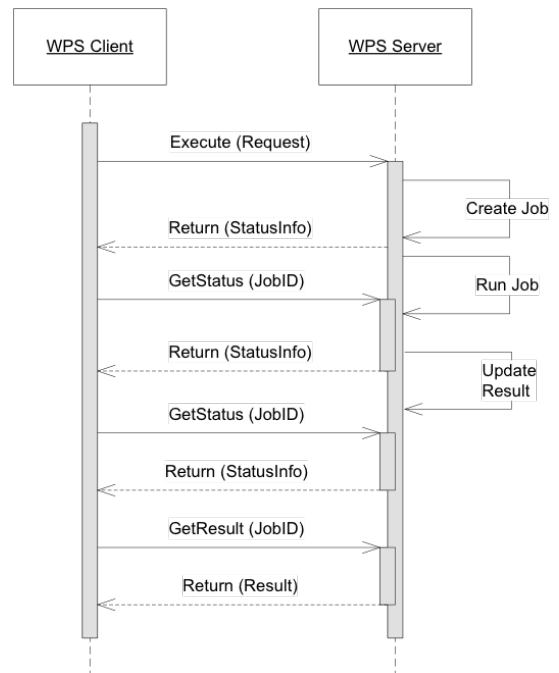
In the synchronous execution model the client performs an Execute operation on the WPS server and waits for the result. The server response is either an error or the execution

result. In this paradigm, the client is blocked until the server finishes the scheduled job and produce a result (or, in many implementation, a timeout occurs). Synchronous execution is the classical client-server interaction and may be the optimal choice for the most of the cases. Figure 3a shows a UML interaction diagram that summarize the client-server interaction using synchronous execution.

The asynchronous execution model is first introduced in the version 2.0 of the WPS standard [28]. The operations involved are: Execute; GetStatus; GetResult. The first operation the client should perform is Execute, specifying the Async execution model. This operation creates and schedules a job on the server. The client receives an identifier of the job as a response. After, the client can continue its calculations/operations as it is not blocked by the server. At regular intervals, the client should perform a GetStatus operation specifying the job identifier. The response include the current status of the job. Eventually, the job finishes and the result becomes available using the GetResult operation in conjunction with the job identifier. Clearly, this execution model is suitable in such situation in which the job takes long time to complete. For example, because the process is particularly time consuming, or because the data size is specially big. Figure 3b shows a UML interaction diagram that clarify the relation between the client and the WPS server.



(a) Interaction diagram for a synchronous execution model



(b) Interaction diagram for an asynchronous execution model

Figure 3: Process execution interaction between a client and a WPS 2.0 compliant server. (a) shows the interaction using synchronous request, while (b) using an asynchronous request. From Figure 3 and 4 of [28]

### 3.3 Issues and improvements

Undoubtedly, WPS 2.0 introduces interesting features over the previous iteration of the standard. Nevertheless, there are some aspects that can be further improved. The following considerations come from the experience earned while developing and studying the subject in the context of this Master thesis.

The first consideration is the data format used for exchanging information between the client and the server. WPS does not enforce the use of XML but encourages it. This is not an issue, rather a limitation. Nowadays, web applications are moving to JSON as data format because of its simplicity and its seamless integration with JavaScript, which is the main language for developing web applications.

Also, the exact format (hierarchy of elements, attributes, et cetera) of the data is not specified into the standard, but is implementation specific. This seems like a paradox giving the aim of the standard is interoperability. Even if the abstract data model is defined, a client still needs to study the specific behavior of a certain server. One server can accept the `mode="async"` as an attribute for the root element of the XML document, while another server may expect it as a XML element per se. Giving a generic (preferred) way of structuring communication data would improve the interoperability of the standard to a real high level. Each supported data format, XML or JSON, can have a template indicating how a client should formulate the requests to a server and how to expect the response.

Finally, the asynchronous execution model implies the use of polling technique for requesting the status of the scheduled process execution. Although there are some attempts to improve the performance of polling [9][18], its efficiency is much less if compared to new techniques like WebSocket [30] and the new features of HTTP2 [17]. The use of these new technologies may lead to higher performance under certain scenarios. Possibly, the support for polling and streaming models would improve the flexibility and adaptability of the WPS standard.



## 4 Web Processing Service Client Library: `async-wps`

This section explains in detail the development of the `async-wps` library.

The motivation for developing `async-wps` comes naturally with the release of WPS standard version 2.0. The new standard offers very interesting features for web services and web applications.

It standardizes the way a client and a server interact in a remote procedure call scenario. Also, the support for asynchronous execution makes the standard useful in performance-sensible scenarios.

Finally, the preferred communication format is XML in WPS. While XML is a very famous data interchange format, easy to read for humans and widely used in the industry, it may not be easy to work with from a programming language. To not worry about object to XML and viceversa conversion, a library is very welcome.

JavaScript is the reference language for developing web applications. While there are attempt of establishing new languages (lastly Dart<sup>2</sup> from Google [8]), no one was widely adopted. JavaScript (formally and implementation of ECMAScript) is already supported by all web browsers. Also, JavaScript has been a growing programming language for server-side scripting. The rise of NodeJS and the support of ECMAScript 5 in the Java 8 version made JavaScript a very appealing programming language.

---

<sup>2</sup><https://www.dartlang.org/>

## 4.1 Current implementations

At the moment, there are two ways to work with a WPS server using JavaScript as a programming language: create an ad-hock implementation; use a library.

The first option has a few advantages and some key disadvantages. The most important advantage is the simplicity of development for a particular use case. This option only applies if the application needs to communicate with a well known WPS server, and a small part of the standard is involved. For example, when a WPS server performs a simple spatial operation between two rectangles. In contrast, the client application needs to create and process XML documents. Using JavaScript it is difficult to work with XML documents. The language specification does not provide API for manipulating XML. In a browser environment (for example, Firefox and Chromium) there are some workaround due to the ability of the browser to work with HTML. In a server environment (for example, NodeJS) the only option is to parse the XML document as a string of character and extract valuable information from it. The client is also in charge of producing valid WPS 2.0 XML documents. This includes the correct position and value of the properties and fields required by the standard. Not complying with the standard specification may lead to unexpected behavior.

The second option to deal with a WPS server is by using a third-party library. This is the most suitable solution for most use cases. A library should be able to communicate with a WPS compliant server while offering a simple API to the user. It should encapsulate (hide) the complexity of the standard and the difficult interaction with XML documents using JavaScript. There are two disadvantages of using a library. In the one hand, sometimes is difficult to find a library that meets the requirements of an application. It may not be available due to license conflict or it may not include all the required features. On the other hand, it is necessary to invest time in learning how the library works. This step is not always easy, specially for small library that are not company-backed. The main barrier may be the lack of documentation and examples. Nevertheless, normally, this kind of libraries are open source and the code can be reviewed by the user and find hint on how the library should be used.

Currently, there are two libraries that allow a user to interact with a WPS server. The first one is wps-js developed by 52°North<sup>3</sup>. The second one is called OpenLayers 2<sup>4</sup>.

#### 4.1.1 52north-wps-js

52north-wps-js is a project started by 52°North company. It is an open source project (Apache Software License 2.0) and it is publicly accessible through the company's Github page<sup>5</sup>.

It provides a way to access WPS 1.0 servers. It provides a standalone JavaScript file to import and a form-based visualization of WPS 1.0 servers. The first is useful when creating web applications, while the latter is useful to try WPS 1.0 servers. It provides the following WPS operations: GetCapabilities; DescribeProcess; Execute.

The library does not offer a direct import option, it is necessary to download and build the Git repository in order to create an "executable" file to use.

The library is tightly coupled with a browser environment. It has hard dependencies with jQuery and OpenLayers 2.

#### 4.1.2 OpenLayers 2

OpenLayers 2 is an open source project (2-clause BSD License) sponsored by OSGEO. The code is accessible from the dedicated Github page<sup>6</sup>.

OpenLayers is a library that allows to integrate dynamic maps into web pages. It supports many OGC standard and, therefore, is able to interact with many geospatial services. Among its features, it is able to access WPS 1.0 servers. Therefore, the operations GetCapabilities, DescribeProcess and Execute are available.

---

<sup>3</sup><http://52north.org/>

<sup>4</sup><http://openlayers.org/two/>

<sup>5</sup><https://github.com/52North/wps-js>

<sup>6</sup><https://github.com/openlayers/ol2>

The library can be included into any web page directly as it is distributed with an already compiled version.

OpenLayers 2 is not the latest version of the library. Version 3.12.1<sup>7</sup> provides many new features and compatibility with new web standards. Although, it does not offer WPS server interaction. For this reason, the OpenLayer 2 is considered for the purpose of this thesis.

---

<sup>7</sup><http://openlayers.org/>

## 4.2 Requirements

Table 22 shows the requirements for the async-wps library.

<b>Requirement ID</b>	<b>Description</b>
R01	Support for WPS 2.0 operations: GetCapabilities; DescribeProcess; Execute; GetStatus; GetResult
R02	Compatibility with NodeJS and browser environment
R03	Support for ComplexData input type
R04	Open source compatible license
R05	JavaScript API for accessing WPS 2.0 servers
R06	Encapsulation of XML to JavaScript Object, and viceversa, conversion

Table 22: Summary of the requirements for the async-wps library

## 4.3 Development and Results

This section describes the development of the `async-wps` library.

### 4.3.1 Technology and design decisions

In order to fulfill the requirement presented in Table 22, this section explains the design and technology decisions.

The programming language of choice for the library is JavaScript. JavaScript is developed incrementally. This means that first, the specification (ECMAScript) is agreed by the ECMA International Organization<sup>8</sup> and, second, developer implements the specification in their platforms. The target platform for this library is NodeJS version 4.2.4 LTS (Long Term Support). This platform is based on the open source JavaScript engine V8<sup>9</sup>, sponsored by Google. This version of NodeJS implements a part of the ECMAScript 2015 (ECMA-262 specification [19]), such as, among others, Classes syntax, Block Scoping variables, Promise API specification, Arrow functions and Template strings. The NodeJS environment supports the asynchronous features that are the key of the strength of `async-wps` library. Mainly, the Promise API and the non-blocking code execution.

The Promise API is the base of the asynchronous capabilities of `async-wps` library. JavaScript has an asynchronous and virtually-parallel execution flow [11], but it is a single-thread language. This can seem a paradox, but in fact it works for most of the situations. The key point is to specify the code execution in a non-blocking way. Non-blocking means that the code flow is not blocked by a function execution, instead multiple code blocks can be set to run at the same time with no need to synchronize the execution. Since JavaScript is single-thread, the execution does not happen simultaneously, but the platform scheduler and compiler may evenly distribute execution time. In this way, overall performance is high. It is important to note that a Promise can end in two states: resolved or rejected. When resolved, the promise is done and, if present, the execution is

---

<sup>8</sup><http://www.ecma-international.org/>

<sup>9</sup><https://developers.google.com/v8/>

passed to the next chained promise (specified using the `.then(...)` method). When a promise is rejected, an error occurred and the error catching code, if present, is executed (specified with `.catch(...)` method).

The compatibility with browser environments, such as Chrome, Safari and Firefox, is achieved using compiling techniques. Using the third-party library Browserify and BabelJS (see Section 4.3.2), the `async-wps` library code is converted in a browser-compatible code. This is necessary since NodeJS uses newer features of JavaScript, while browsers are more conservative and they must maintain compatibility to older standards in order to not break existing web pages code.

The library `async-wps` can be imported into bigger projects using a variety of methods. It needs to support ECMAScript 2015 Modules, AMD and CommonJS/Node. External modules import in standard JavaScript has been introduced in ECMAScript 2015. This lack of functionality in the previous versions, created different implementation of the "import" as third-party libraries. Since there was no standard, each library implemented the functionality in a different way. While the implementation is different, the philosophy remains the same, to provide the ability to load external JavaScript modules. In the NodeJS environment, the library should be imported using the NodeJS import system, CommonJS. In a browser environment, the library can be loaded using standard HTML Script tag, the new ECMAScript 2015 modules syntax and RequireJS. This is needed because `async-wps` needs to be able to be imported in any project, independently of the used import system.

The XML document management complexity is hidden to the final user. The `async-wps` library allows users to interact with its API using JavaScript objects. Internally, these objects are converted to XML document for the interaction with the WPS 2.0 compliant server. Also, the server response XML document is parsed and converted to JavaScript object and then offered to the user of the library.

The license for the project is GPL 3.0<sup>10</sup>. This license is sponsored by the Free Software

---

<sup>10</sup><http://www.gnu.org/licenses/gpl-3.0.en.html>

Foundation and it is compatible with the open source and free software philosophy.

The source code and tests are developed with version 4.2.4 of NodeJS as target platform. This offers a good environment for the development and test of the system. It has much more flexibility than a browser environment and every phase of the development can be achieved in a IDE. For this purpose, WebStorm 10 from JetBrains<sup>11</sup>. Some of the most important feature of this IDE are:

- Support for NodeJS environment
- Smart code completion for JavaScript
- Support for Mocha BDD testing
- Integrated debugging tools
- Code quality tools

### 4.3.2 Requirements fulfillment

This section covers the requirements for the library and how are they solved.

#### **Requirement R01**

The Listing 1 shows a sample code in JavaScript that illustrate the use of `async-wps` API. This example shows an interaction between a client that uses the `async-wps` library and a WPS 2.0 server. The code makes use of the Promise API and shows the integration with the five supported operations: `GetCapabilities`; `DescribeProcess`; `Execute`; `GetStatus`; `GetResult`.

---

<sup>11</sup><https://www.jetbrains.com/webstorm/>



---

```
1 function processCapabilities(...) { ; }
2 function processDescription(...) { ; }
3 function waitForCompletion(...) { ; }
4 function processResult(...) { ; }
5 function logError(...) { ; }
6
7 let getUrlFromForm = new Promise((resolve, reject) => {
8   let url = document.getElementById('url-input').value;
9   if( ! isValid(url) ) reject('URL is not valid');
10  else resolve(url);
11 });
12
13 getUrlFromForm
14   .then(wpsjs.GetCapabilities.executeGet)
15   .then(processCapabilities)
16   .then(wpsjs.DescribeProcess.executePost)
17   .then(processDescription)
18   .then(wpsjs.Execute.executePost)
19   .then(waitForCompletion)
20   .then(wpsjs.GetResult.executePost)
21   .then(processResult)
22   .catch(logError);
23
24 // The code below does not wait for the algorithm to be done to execute
25 ...
```

---

Listing 1: Sample code for a interaction with a WPS 2.0 server in JavaScript. This code makes use of the Promise API found in the ECMAScript 2015 specification

## Requirement R02

The library is developed with NodeJS as target platform. The support for a browser environment is achieved using two third-party libraries: Browserify<sup>12</sup> and BabelJS<sup>13</sup>.

Browserify is a JavaScript library that is able to pack a NodeJS projects in a way that is compatible with web browsers. The dependencies of the project are bundled together and a single JavaScript file is created. Also, Browserify provides a browser-compatible implementation of the NodeJS base library, for example the `require` function or the `http` NodeJS native module. In this way, a code that makes use of the `http` NodeJS module is able to run on a browser environment without changes. Furthermore, Browserify is able to compact and optimize the code.

BabelJS is a JavaScript compiler that makes JavaScript code backward compatible with the desired JavaScript version. In order to be able to run code that makes use of new features in older JavaScript environments code modifications are needed. Browserify does this modifications automatically. It has implementations of the latest ECMAScript features for previous version of the language. For example, the Promise API is introduced in ECMAScript 2015. This means that a older JavaScript environment does not support it natively. In this case, Browserify provides a module that introduces this new feature, so that the developer does not have to modify the code.

## Requirement R03

ComplexData is a supported data type for inputs element in Execute operations. Code Listing 2 shows an example on how to use a ComplexData input type in a asynchronous Execute operation. As shown in line 6, `async-wps` offers a constant to specify a Complex-Type input (`wps.Constants.COMPLEX_TYPE`) and a `content` property for specifying the desired data.

## Requirement R04

---

<sup>12</sup><http://browserify.org/>

<sup>13</sup><https://babeljs.io/>

---

```
1 var params = {
2     mode: 'async',
3     identifier: 'process-id',
4     inputs: [
5         {
6             type: wpsjs.Constants.COMPLEX_TYPE,
7             identifier: 'complex-input',
8             mimeType: 'text/xml',
9             encoding: 'utf-8',
10            content: '<complexData>...</complexData>'
11        }
12    ]
13 };
```

---

Listing 2: Sample code for the specification of a ComplexData input for an Async process execution

The license of the project is GPL 3.0. This license is fully open source and is sponsored by the Free Software Foundation.

### Requirement R05

The API of the library contains five objects. Each object represents a WPS 2.0 operation: `GetCapabilities`; `DescribeProcess`; `Execute`; `GetStatus`; `GetResult`. The library allow the operations to be executed using a HTTP GET or HTTP POST request. Therefore, each object provides two different methods:

- `executeGet(url)`: it executes a GET request to the specified URL. The parameters of the operation are specified using Query Parameters in the URL itself.
- `executePost(url, parameters)`: it executes a POST request to the specified URL. In this case, the parameters are provided using a JavaScript object. The properties of the object are extracted and a XML document is created. Finally, the document is attached to the request using the body of the POST request.

Each method return a JavaScript Promise object. When the promise resolves, the callback will receive a object with three properties:

- **parsedData**: a JavaScript object that represents the response XML document maintaining the hierarchy of the attributes and properties.
- **rawData**: string property with the full response XML document as retrieved from the server.
- **data**: a JavaScript object that provides simpler access to some meaningful properties of the response XML document. Also, it can provide aggregated data and direct access to important properties that are deep in the hierarchy of the XML document.

### **Requirement R06**

The XML document manipulation required by the WPS standard is hidden to the user. The user specifies and receives input and outputs using JavaScript objects. Internally, the library creates and parse XML documents that need to be sent and are received from a WPS 2.0 server.

The transformation from a XML document to a JavaScript object is done using the jsonix third-party library. This library uses predefined schemas to map a XML document to a JavaScript object. Schemas for all the OGC standards (including WPS 2.0) are available<sup>14</sup>. When a XML document is received, jsonix first identify the correctness of it according to the standard, then maps its properties to a JavaScript object.

In order to be able to convert a JavaScript object to a valid WPS 2.0 XML document, templates are used. This technique uses some predefined texts with some special symbols that are used as placeholders for dynamic data. The `async-wps` library uses `lodash` template strings to achieve this goal. Code Listing 3 shows the template string for creating a `GetStatus` request. The function `_.template('')` creates an "executable" version of

---

<sup>14</sup><https://github.com/highsource/ogc-schemas>

---

```
1 var template = _.template(`
2     <wps:GetStatus service="WPS" version="2.0.0"
3         xmlns:wps="http://www.opengis.net/wps/2.0"
4         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5         xsi:schemaLocation="http://www.opengis.net/wps/2.0 ../wps.xsd ">
6         <wps:JobID><%= jobID %></wps:JobID>
7     </wps:GetStatus>
8 `)
```

---

Listing 3: Template string for a GetStatus request document

the template to which it is possible to pass parameters. In the showed example, the only necessary parameter is the jobID.

### 4.3.3 Dependencies

This section provides a brief description of the library that are needed at runtime for `async-wps` to execute. All the libraries are open source and in the case of a browser-compatible build, they are bundled with `async-wps`. Mocha and Chai are testing dependencies. They are not required for a normal execution of the library but are used to run the tests.

**jsonix** <sup>15</sup> This library provides a mapping between JavaScript objects and XML documents. It is able to make the conversion in both directions. Also, it is able to validate a XML document with a XMLSchema. To make a conversion, this library needs a mapping file in which is specified the rules for converting the XML document to a JavaScript object and vice versa. Mappings for any OGC standards are available online.

---

<sup>15</sup><https://github.com/highsource/jsonix>

**lodash** <sup>16</sup> This library includes many utility functions to the JavaScript language. It has interesting functions for functional programming such as: map; reduce; pick; plunk; any; some. Also, it provides template string for creating text from predefined templates and dynamic data.

**mocha** <sup>17</sup> This is a testing framework for JavaScript. Its strengths are in the simplicity of the definition of tests and the asynchronous execution of them.

**chai** <sup>18</sup> This library provides fluid assertion functions. Its primary goal is to be used in conjunction with a testing framework. It provides both BDD and TDD styles assertions.

#### 4.3.4 Tests

The `async-wps` library code comes with unit testing to improve code quality and robustness. The tests are executed in a NodeJS environment using Mocha and Chai third-party libraries.

A very simple testing WPS 2.0 server is provided with the library. It is implemented in NodeJS. This server expects and returns valid WPS 2.0 XML documents.

The `async-wps` tests are intended to test the library functionality against the XML documents provided in Annex B of WPS 2.0 standard document [28].

#### 4.3.5 Library compilation and test

The library `async-wps` ships with a `build` folder in which is possible to find compiled and ready-to-use version of the library. Code Listing 4 shows the steps to build and test

---

<sup>16</sup><https://lodash.com/>

<sup>17</sup><https://mochajs.org/>

<sup>18</sup><http://chaijs.com/>

---

```
1 git clone https://github.com/andreacalia/wps-js.git
2 npm install
3 npm run test
4 npm run build-all
```

---

Listing 4: List of console commands to build and test the async-wps library

manually the library (the commands are shown on a standard UNIX console system).

#### 4.3.6 Source code

The VCS for this project is Git<sup>19</sup>. Git is a distributed version control system that aims to be fast and easy to learn. It can manage projects of every size and it can be fully integrated with Github. Github<sup>20</sup> is a web application that manages software projects that use Git as VCS. It supports collaboration, code review and more features that allows easy management of big projects. Github offers free hosting for open source projects. For this reason, the code of async-wps is hosted on Github and can be found here: <https://github.com/andreacalia/wps-js>.

---

<sup>19</sup><https://git-scm.com/>

<sup>20</sup><https://github.com/>

## 4.4 Limitations

Current implementation of the WPS 2.0 standard in `async-wps` is not complete. Most of the features the standard provides are already present in the library, as well as the API structure.

A current limitation can be found in the automatic management of XML documents. These documents can be rather complicated and includes portions of other OGC standards. The support for every possible combination of OGC standards is complicated to achieve. Yet, the modular structure of `async-wps` allows easy integration.

Another limitation is the support for only `ComplexData` input types. WPS 2.0 standard includes other two types of input type: `LiteralData` and `BoundingBox`. They are not currently included in the library. Placeholders are included and the library is already able to identify them, throwing a "not implemented" exception.

These limitations do not affect the usability of the library for the sake of this thesis. The main goal of this thesis is to provide a starting point for a fully fledged WPS 2.0 support in JavaScript. This base needs to be modular, robust yet flexible and well tested. Those are the main focuses of the development.



## 5 Case Study: Indoor Location

This section explains the case study for this thesis. Indoor Location is a hot topic now a days and it presents some interesting academic and business-related challenges. It can be used in a variety of situations. From security to advertisement. For example, it can be used to efficiently evacuate people from a crowded museum in case of emergency. Another example is to offer special prices to customers based on their location on the mall.

Section 5.1 explains the definition of Indoor Location. Section 5.2 contains a brief introduction to the machine learning classification method and which is the scope of classification in the Indoor Location context taken into account in this thesis. Section 5.3 gives details of the WPS 2.0 server developed on the context of the thesis. Finally, Section 5.4 explains in details the tool created to manage Indoor Location data and how WPS 2.0 is used to communicate to the data-processing server.

## 5.1 Definition

Indoor Location is a generic term that refers to the spatial localization of an entity in a closed environment. Usually, the entity is a user of the system (a person), but it can also be a thing (sensor, printer, PC, robot, etc.) in a IoT environment.

Classical entity positioning is achieved using GNSS systems, such as GPS (USA), GLONASS (Russia) and Galileo (Europe). These systems use satellites to calculate the position of an entity on the Earth surface. This is achieved using a combination of satellite to triangulate the position of the entity given the known position of the satellites. An entity in this system needs to be in a more or less open environment, because it is needed to have a view of the satellite constellation. This can be a problem on, for example, subways, tunnels, thick forests, small alleys of cities and building interiors. In these situations, the satellites signal intensity is not sufficient to be able to determine the position of the entity in reliable way.

Indoor Location is a localization system that aims to locate an entity in a building or, more broadly, in an environment in which classical GNSS does not work properly. There are a large number of techniques to achieve this goal. According to [4], there are two main distinction of indoor positioning system: infrastructure-based; infrastructure-less. Infrastructure-based indoor positioning systems use a combination of sensors to produce and receive a specific signal that is then processed and analyzed. Such systems includes RFID[25], infrared[36], Bluetooth[32] and ultrasound[31][23] sensors. Infrastructure-less indoor positioning systems make use of the signal that are already present in a specific area to locate entities. Normally, these signals are portions of the electromagnetic spectrum and can be, among others, Wi-Fi[39][20][3], FM radio[4] and magnetic field[6].

A comparison of the most used Indoor Location techniques has been made for the occasion of the Microsoft Indoor Localization Competition [22].

## 5.2 Classification problem

In the context of this thesis, the Indoor Location is considered a classification problem. The case study is based on the supervised creation of a database with samples that indicates various sensors value in a specified location in the space. Sensors values  $S_1, S_2, \dots, S_i$  and positioning data  $L_1, L_2, \dots, L_j$  represent a sample in the database. The sensors values are may be the intensity signals of Wi-Fi access points, FM radio signal or whatever sensor the system may include. Positioning data refers to meta-data that identify the location of the sample, it may include floor and building number, latitude and longitude, zone id, etc.

Given a database in the described form (as a training set) and a new sample, the Indoor Location system should be able to locate the sample in the space. For example, given  $S_1, S_2, \dots, S_i$  a simple system may be able to identify only the building id of an unknown sample. More elaborated systems may even be able to estimate latitude and longitude.

In order to be able to process and identify an unknown sample, a classification algorithm is executed. One of the most used and simpler algorithm is kNN[7]. In summary, this algorithm is able to determine the class of a sample based on the nearest class in the database using a specific distance measure. Number of voting classes, distance function and other parameters are able to adapt the algorithm behavior to the specific problem. In the context of this application the classes are the different values of the positioning data. In a simple example, the system may be able to identify the building id of a unknown sample. In this situation, the classes are the different building ids that are present in the database. The distance measure is an important factor for the performance of the system. A good distance measure heavily depends on the database used[34].

### 5.3 WPS 2.0 server: wps-classifier

This section explains in details the implementation of the WPS 2.0 server for classifying data using the kNN machine learning classification algorithm. The name of this Software is wps-classifier.

This server implements the WPS standard version 2.0. This means that provides the implementations for the following operations: GetCapabilities; DescribeProcess; Execute; GetStatus; GetResponse.

Classification algorithm such as kNN can be time consuming depending on the size of the data. Also, in a web scenario, the client can vary from a Smartphone to a super computer. For this reason, delegating the classification processing to a server increase the overall performance of the system. This is specially true when the data size is specially high. In this case, the ability of WPS 2.0 to work in asynchronous mode leads to even more, raw or perceived, performance in the client side.

WPS 2.0 supports two execution models, synchronous and asynchronous. wps-classifier supports both execution models for providing flexibility to the user. In this way, it can choose which execution model fits its needs for a specific problem.

#### 5.3.1 Requirements

Table 23 shows the requirements for the wps-classifier web service.

Requirement ID	Description
R01	WPS 2.0 compatible web API
R02	Support for asynchronous and synchronous process execution models
R03	Support for large quantity of input data

Table 23: Summary of the requirements for the wps-classifier

## 5.3.2 Development and Results

This section describes the development of the WPS 2.0 wps-classifier server.

### 5.3.2.1 Technology and design decisions

In order to implement the requirements specified in the Table 23, the following technology and design decisions are made.

The API of the server is composed by a single entry point. The URL is in the form: `http://localhost:4567/wps`. This decision allows the user to not be confused about URLs of the server and it is consistent with the WPS standard. To access WPS operations, the user connects to the entry point using the HTTP POST method. In this way, the WPS payload is included in the **body** of the HTTP request and is then extracted and processed by the server. Based on the content of the payload, the different WPS operations are performed.

The asynchronous execution model implies the storage of some information about the process executions. This information includes the current status of the process and, eventually, the result or the error that occurred. This information is stored in a in-memory Redis database. This database provide a simple key-value storage that is very suitable for this scenario.

In order to be able to support large input files, the server implements the possibility to upload the WPS payload using raw HTTP POST body or using a XML file. The second option is specially suitable for web applications in which the concatenation of large data in a, for example, AJAX request can affect the performance.

The language of choice for developing wps-classifier is Java version 8. Java is a largely established programming language for developing web services due to its scalability, robustness and type safety. The Java ecosystem is huge and it has tools that ease the development of such Software.

### 5.3.2.2 Requirement fulfillment

This section explain in details the implementation of the requirements for the wps-classificator Software.

#### **Requirement R01**

In order to be fully compatible with WPS 2.0, the server have to implement the following operations: GetCapabilities; DescribeProcess; Execute; GetStatus; GetResult.

GetCapabilities gives a description of the WPS server. It includes the name, abstract, version and the processes it contains. Listing 5 shows a short (for brevity) version of the XML document returned by the server.

DescribeProcess operation gives a detailed description of a process available on the server. Since the server supports only the kNN classification for now, this is the only returned description. Listing 6 shows the XML document returned by the server.

Execute operation is supported for both asynchronous and synchronous execution models. The result may be a StatusInfo document (Listing 8) or a Result document (Listing 9) for, respectively, asynchronous and synchronous mode. More details about the implementation of this operation can be found in the description of the Requirement R02.

GetStatus is used to retrieve the status of a scheduled process. the status can be: Succeeded; Accepted; Running; Failed. Refers to Section 3.1.4 for a detailed description of the meaning of such statuses. Listing 8 shows the XML document returned for a GetStatus operation for the scheduled process with id 8132019285681.

GetResult operation gets the result of a scheduled process on the server. Given the id of the process, a Result document is returned with the result. Listing 9 shows the result of the process with id 8132019285681.

---

```

1 <wps:Capabilities service="WPS" version="2.0.0">
2   <ows:ServiceIdentification>
3     <ows:Title>WPS kNN Classification server</ows:Title>
4     <ows:Abstract>This server provides kNN classification machine
5       learning algorithm.</ows:Abstract>
6     <ows:Keywords>
7       <ows:Keyword>Geoprocessing</ows:Keyword>
8       <ows:Keyword>kNN</ows:Keyword>
9       <ows:Keyword>classification</ows:Keyword>
10    </ows:Keywords>
11    <ows:ServiceType>WPS</ows:ServiceType>
12    <ows:ServiceTypeVersion>2.0.0</ows:ServiceTypeVersion>
13    <ows:Fees>NONE</ows:Fees>
14    <ows:AccessConstraints>NONE</ows:AccessConstraints>
15  </ows:ServiceIdentification>
16  <ows:OperationsMetadata> (removed for brevity)
17  </ows:OperationsMetadata>
18  <wps:Contents>
19    <wps:ProcessSummary jobControlOptions="sync-execute
20      async-execute">
21      <ows:Title>kNN classifier</ows:Title>
22      <ows:Identifier>kNN</ows:Identifier>
23    </wps:ProcessSummary>
24  </wps:Contents>
25 </wps:Capabilities>

```

---

Listing 5: Sample XML document for a GetCapabilities operation of the wps-classifier

---

```

1 <wps:ProcessOfferings>
2   <wps:ProcessOffering jobControlOptions="sync-execute async-execute"
3     outputTransmission="value">
4     <wps:Process>
5       <ows:Title>kNN classification</ows:Title>
6       <ows:Identifier>kNN</ows:Identifier>
7       <wps:Input>
8         <ows:Title>trainData</ows:Title>
9         <ows:Identifier>trainData</ows:Identifier>
10        <wps:ComplexData> <wps:Format
11          mimeType="application/json"encoding="UTF-8"/>
12        </wps:ComplexData>
13      </wps:Input>
14      <wps:Input>
15        <ows:Title>testData</ows:Title>
16        <ows:Identifier>testData</ows:Identifier>
17        <wps:ComplexData> <wps:Format
18          mimeType="application/json"encoding="UTF-8"/>
19        </wps:ComplexData>
20      </wps:Input>
21      <wps:Output>
22        <ows:Title>Evaluation of evaluating the model</ows:Title>
23        <ows:Identifier>1</ows:Identifier>
24        <wps:ComplexData> <wps:Format mimeType="text/plain"
25          encoding="UTF-8"/>
26        </wps:ComplexData>
27      </wps:Output>
28    </wps:Process>
29  </wps:ProcessOffering>
30 </wps:ProcessOfferings>

```

---

Listing 6: Sample XML document for a DescribeProcess operation of the wps-classificator for describing the kNN process



---

```
1 <wps:Execute version="2.0.0" response="document" mode="async">
2   <ows:Identifier>kNN</ows:Identifier>
3   <wps:Input id="trainData">
4     <wps:Data>
5       <wps:ComplexData>
6         {
7           "data": "...",
8           "classIndex": "BUILDINGID"
9         }
10      </wps:ComplexData>
11    </wps:Data>
12  </wps:Input>
13  <wps:Input id="testData">
14    <wps:Data>
15      <wps:ComplexData>
16        {
17          "data": "...",
18          "classIndex": "BUILDINGID"
19        }
20      </wps:ComplexData>
21    </wps:Data>
22  </wps:Input>
23 </wps:Execute>
```

---

Listing 7: Sample XML document for the execution of the kNN classification with the specified input data

## Requirement R02

The server wps-classificator supports both synchronous and asynchronous execution models.

During a asynchronous execution, a token is created for the scheduled process and a StatusInfo (Listing 8) document is returned immediately. Behind the scene, the server starts a Thread that executes the operation with the given inputs. The Thread updates the status of the scheduled process on the Redis database. When it terminates, the result is stored on the database and the status is set to **Succeeded**. Meanwhile, the client can query for the status of the scheduled process, as its status is retrieved from the database. Finally, the client asks for the results of the process, Result document 9.

In contrast, the synchronous execution does not start a Thread, instead the process execution is done during the request. The result of the process is sent as the response of the execution request (Listing 9).

---

```

1 <wps:StatusInfo xmlns:ows="http://www.opengis.net/ows/2.0"
2   xmlns:wps="http://www.opengis.net/wps/2.0"
3   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4   xsi:schemaLocation="http://www.opengis.net/wps/2.0 ../wps.xsd">
5   <wps:JobID>8132019285681</wps:JobID>
6   <wps:Status>Accepted</wps:Status>
7   <wps:NextPoll>2016-01-02T12:23:08.230Z</wps:NextPoll>
8 </wps:StatusInfo>

```

---

Listing 8: Response of an asynchronous WPS execute request or a GetStatus request. It includes the JobId that can be used to query the status and result of the processing

---

```

1 <wps:Result>
2   <wps:JobID>8282212079926</wps:JobID>
3   <wps:ExpirationDate>2016-01-02T12:25:28.467Z</wps:ExpirationDate>
4   <wps:Output id="1">
5     <wps>Data>
6     Correlation coefficient          0.5602
7     Mean absolute error             0.7143
8     Root mean squared error        1.8898
9     Relative absolute error        51.6129 %
10    Root relative squared error     85.1691 %
11    Total Number of Instances      7
12   </wps>Data>
13   </wps:Output>
14 </wps:Result>

```

---

Listing 9: Response of a synchronous WPS execute request or a GetResult request (assuming the result is ready). It includes the results of the processing

### Requirement R03

In order to be able to easily send large quantities of data as input for the WPS execute operation, wps-classificator supports file upload or HTTP POST body.

File upload is suitable when the client already have the WPS payload codified as a file on the disk and this file is very large. The HTTP header `multipart/form-data` will speedup the performance of sending large files over HTTP.

It is also possible to include the WPS payload in the body of a HTTP POST request. In this case, the payload is sent and codified as normal string and is embedded in the HTTP request. This method is suitable when the payload is relatively small in size. It is a perfect option for request the operations: `GetCapabilities`; `DescribeProcess`; `GetStatus`; `GetResult`.

#### 5.3.2.3 Dependencies

This section provides a brief description of the wps-classificator dependencies. All the third-party libraries described below are open source.

**Java Spark** <sup>21</sup>Spark Java is a microframework for developing web applications using the Java programming language. Spark Java is very simple to learn and provides high performance and scalability. It is not based on the Java EE stack and it fully supports Java 8 features.

**Redis** <sup>22</sup>Redis is a in-memory (but persistent) database that is able to achieve high performance when dealing with large quantities of data. This is largely due to the fact that it stores data in a simple key-value pairs.

---

<sup>21</sup><http://sparkjava.com/>

<sup>22</sup><http://redis.io/>

### 5.3.3 Source code

Git is used for managing this project. As remote repository, Bitbucket is used. Bitbucket is a web platform for managing Git projects. The source code for this project can be found here: <https://bitbucket.org/andreacalia/wps-classification-server>.

## 5.4 Web Application: Indoor Location Management Tool

This section explains in details the tool that is created during the thesis to be able to manage Indoor Location databases (in specific [35]). This tool is a web application and simplifies the operations of analysis and process of the data. It connects to wps-classificator to perform expensive analysis tasks.

Indoor Location databases are difficult to manage and analyze. The characteristic of this database is that they contains numeric and spatial information.

The numeric information is composed by the sensors values, while the spatial information may include coordinates (latitude and longitude for example), space ids, etc.

Normally, a table-based visualization and filtering is suitable for databases that does not include spatial information. In the particular scenario of Indoor Location databases, along with classical table-based visualization, it may be more convenient to have a map-based visualization. In this way, some GIS analysis may be applied on the data and it may be easier to identify meaningful data.

Also, the proposed tool as a web application is interesting due to its cross-platform nature. Scientists may use their preferred software stack (OS X, Linux or Windows) and they can continue to work with the tool.

Finally, the classification of samples and the data analysis it is shifted to a distributed web service and this can drastically improve the overall performance of the system.

### 5.4.1 Requirements

Table 24 shows the requirements for the Indoor Location Management Tool.

Requirement ID	Description
R01	Support for data visualization using tabular and map view

R02	Support for data filtering using for queries and geospatial operations
R03	Support for the creation of data clusters for performing periodical analysis
R04	Open source compatible license

Table 24: Summary of the requirements for the Indoor Location Management Tool

## 5.4.2 Development and Results

This section describes the development of the Indoor Location Management Tool.

### 5.4.2.1 Technology and design decisions

In order to fulfill the requirements presented in Table 24, this section explains in details the design and technology decisions.

The first design decision for the tool is the way it loads a indoor location database. It can be done in two ways: via CSV file or via fetching from a web service. This first option involves the load of a CSV file from the user computer. This file should contain a row with the column header and

Indoor Location Management Tool is a web application. Therefore, it is developed using HTML/CSS and JavaScript. Although, new versions of modern web browsers feature most of HTML/CSS and JavaScript standards, the implementation may slightly vary. Therefore, a cross-browser third-party library is needed. This kind of libraries are of two types: custom API or polyfill. When providing a custom API, the library forces the developer to learn and use their API to make standard operations. These operations are then performed in different ways depending on the browser in which they are running. In contrast, a polyfill library provides an implementation of a standard feature and it is used when a native implementation is not present.

The requirements of the tool make the UI of the system quite elaborated. There is the necessity of visualizing the data using both map and tabular view. The creation of a mixed view layout of this complexity can be challenging. For this reason a UI framework that provides a MV\* (ModelView) design pattern is desirable. This framework should provide a basic structure for data models and a representation of them using views. Also, a mechanism for creating complex layouts involving nested views.

The tabular view of the data should be presented using well-known tables. Due to the possible length of the database, a pagination control is required. In this way, only a certain amount of data is shown on screen and the other parts are reachable using pages controls.

The map view should present a map widget on screen. This widget should include all the controls for zooming, panning and identify the feature on the map. Each element of the database, which includes positioning data, is shown on the map and a preview of its sensors data should be available with a click using a info window. Due to the possibly high size and concentration of database elements, the map should be able to show the database using clustering. Clustering technique scans the data and finds the elements that will be shown so close that are not easily distinguishable. For these elements, a cluster is shown which provides a summary of the information on the included elements. This technique allows a much clearer view of crowded data at all levels of zoom.

Both tabular and map view should include filtering controls. These controls are able to select data based on the properties of a database element. For alphanumerical properties, such as sensors values, the tabular view provides a classical form to build a query. This form allows the user to dynamically create a query by comparing properties with specific values. For example, a user may want to select all the elements that have the word "*wifi*" in the property "*title*". In contrast, for geospatial properties, the map view should provide filtering options based on geospatial operations. For example, a user may want to select all the elements of the database that are within a certain polygon or that are close to a specific point of interest.



The filters create selection of element of the database. The selection can be saved for a later use. Saving a selection will create an experiment. An experiment is a set of element of the database that are grouped together for analysis reasons. For example, a user may want to create an experiment with those elements that have a value greater than 5 for the sensor  $S_5$ . Along with a set of elements, an experiment have a name and a description. It is important to note that an element of the database can pertain to multiple experiments.

The experiments are used to perform classification analysis using a WPS 2.0 server. A experiment may be used as a training or testing data for the classifier. In this way, a user may be able to use a subset of the database elements (an experiment) as simulated users in the system and then test its performance.

#### **5.4.2.2 Requirement fulfillment**

This section explains how the requirements are implemented in the Software.

##### **Requirement R01**

The tool provides tabular and map visualization for the elements of the database.

The tabular view, Figure 4, is composed by a table and pagination controls. The number element shown on the page can be configured using the configuration page. Based on the number of elements per page and the total number of elements in the database, the tool calculates the necessary pages needed to visualize all the data. The pagination controls allow the navigation between the pages.

The map view offers the visualization of the elements on a map, Figure 5. If present, the coordinates of a database element is assumed to be in the WGS84 latitude, longitude coordinate reference system. This allow the correct positioning and aligning of the database elements with a basemap or custom geospatial layers. The maps controls allow the user to zoom, pan and identify the elements. They are clustered if considered too close. The minimum distance threshold can be configured in the configuration page.

## Observations

Id	Longitude	Latitude	Floor	Buildingid	Spaceid	Relativeposition	Userid	Phoneid	Timestamp
2497	-0.06608016392800264	39.991881148086826	3	2	233	2	6	19	1371714719
2498	-0.06607711235096252	39.99186546880076	3	2	234	2	6	19	1371714737
2499	-0.06606857296588754	39.99187622172737	3	2	235	2	6	19	1371714754
2500	-0.06601627574499848	39.99183996294016	3	2	238	2	6	19	1371714810
2501	-0.06602749390625141	39.991844733761866	3	2	236	2	6	19	1371714775
2502	-0.06601970191946138	39.99185544815079	3	2	237	2	6	19	1371714794
2503	-0.06598097375096278	39.99179192435328	3	2	241	2	6	19	1371714872
2504	-0.06597417350423494	39.991801311839104	3	2	242	2	6	19	1371714888
2505	-0.06594725638507358	39.991838486683214	3	2	243	2	6	19	1371714907
2506	-0.0660001967997178	39.991781391647706	3	2	240	2	6	19	1371714857

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

Figure 4: The tabular view visualizes the element of the database in rows. Each column represent a attribute of the element. Pagination is added because the expected number of entries of the database is large. The page size can be customized

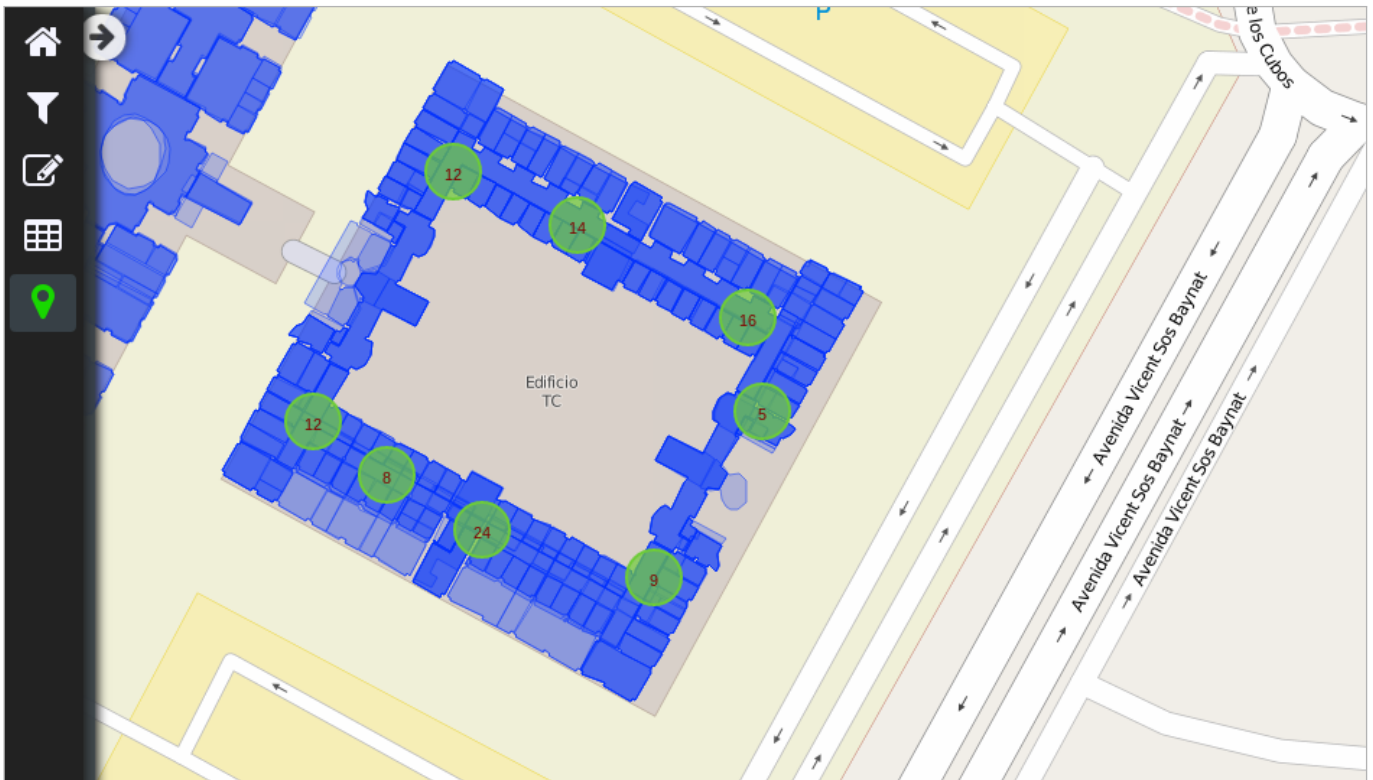
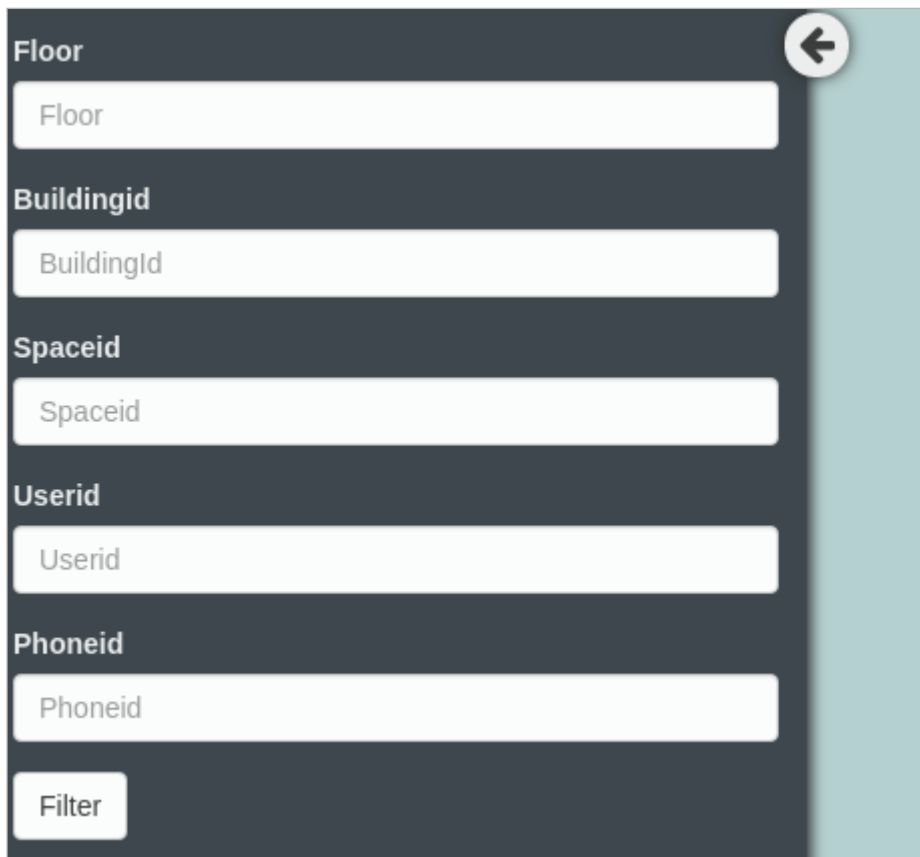


Figure 5: The map view visualize the data taking into account the geospatial attributes of the elements. Those are latitude and longitude specified using WGS84. Custom layers can be added into the map to better understand and analyze the data. Elements that are very close to each other in the selected zoom are clustered together for clarity reasons

## Requirement R02

The filtering using the tabular view can be done using a form (Figure 6). This is a query builder and allows a user to specify the query that is then used to select and filter the elements of the database.

The filtering option using the map view is based on a geospatial operation (Figure 7). In this case, the user can draw a polygon and the elements (features) that are within the polygon are added to the selection.



The image shows a mobile application interface for filtering data. The interface is dark-themed with white text and input fields. It features five filter categories: 'Floor', 'Buildingid', 'Spaceid', 'Userid', and 'Phoneid'. Each category has a corresponding white input field. At the bottom, there is a 'Filter' button. A back arrow icon is visible in the top right corner.

Figure 6: Filtering options using a form. It is possible to filter the data based on non-geospatial properties. In this early version, the filter options occurs comparing using equality the properties set in the form with the ones stored in the database

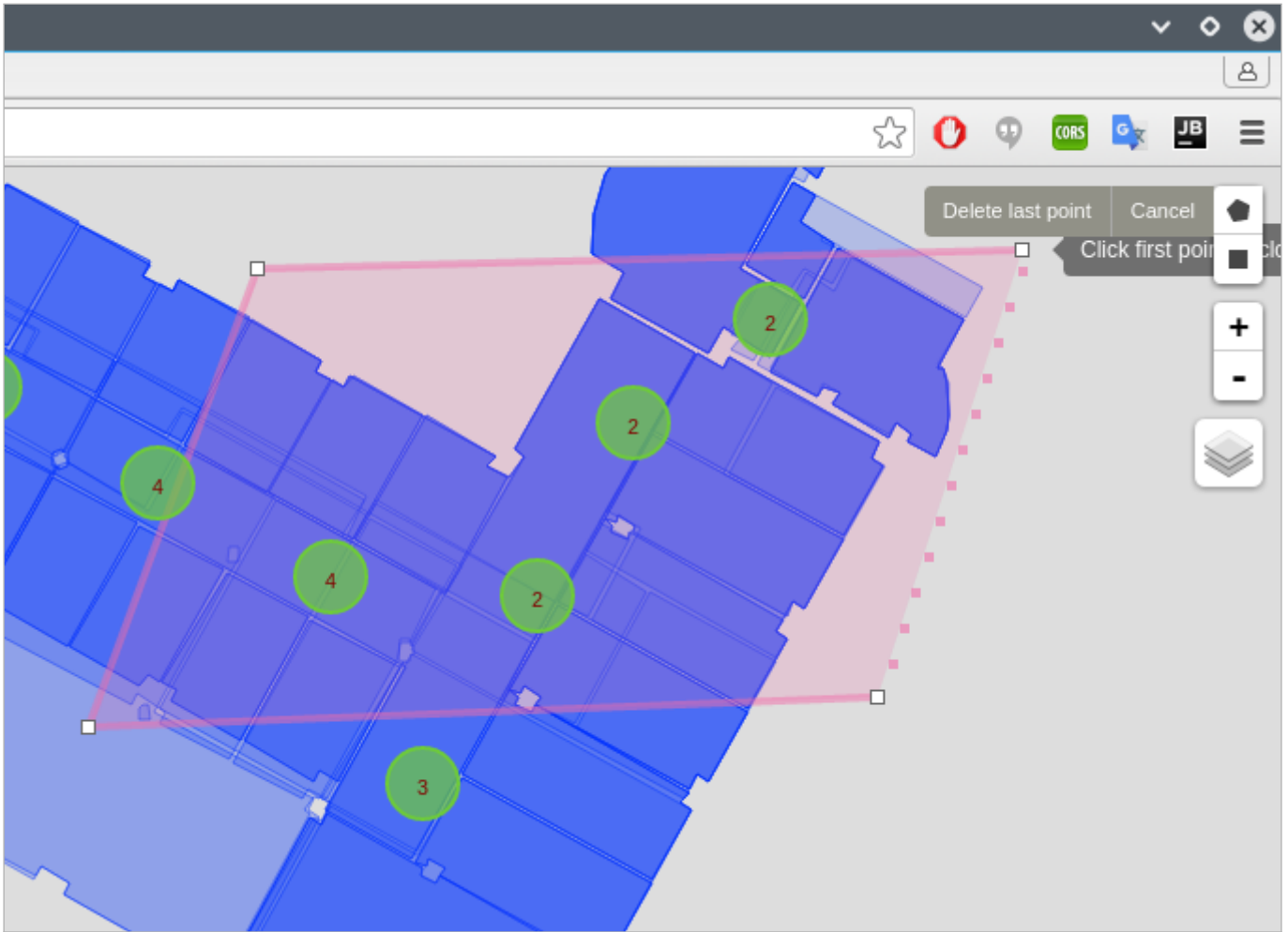


Figure 7: Filtering options for geospatial attributes. The filter happens using the geospatial operation "within". The user chooses the preferred method (arbitrary polygon or rectangle), draws the feature on the map and the database entries that fall into the feature are selected automatically

### Requirement R03

The selection that a user performs can be saved in the application and is called "experiment" (Figure 8). A user can use an experiment for analysis, modification or deletion. A database element may be included in more than one experiment. An experiment has a unique name and a description. These metadata can be used for storing meaningful information about the purpose of the experiment.



Figure 8: The experiment details are shown in the sidebar of the web application. In the central part, the elements of the experiment are visualized

### Requirement R04

The license of the project is Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International<sup>23</sup>. This license is open source. Modification and distribution of the software are allowed given it is not used for commercial purposes, it has proper attributions and it is shared with a compatible license.

<sup>23</sup><https://creativecommons.org/licenses/by-nc-sa/4.0/>

### 5.4.2.3 Dependencies

This section provides a brief description of the Indoor Location Management Tool dependencies. All the third-party libraries described below are open source and are loaded by the application during the startup.

**Bootstrap** <sup>24</sup>This is a popular front-end framework for developing web applications. It provides basic structure for building responsive and high quality web pages. It also features widget and plugins to ease the development.

**jQuery** <sup>25</sup>This is a very famous JavaScript library. It provides a cross-browser API for, among others, DOM manipulating, Ajax and event handling. It is also extendible with a plugin system to include more functionality.

**BackboneJS** <sup>26</sup>This is a lightweight JavaScript library that provides a simple model-view system for making user interfaces in web application. It also features a collection system for the models and a custom event system for UI interactions.

**MarionetteJS** <sup>27</sup>This is a complementary framework of BackboneJS. It extends BackboneJS and provides opinionated code structure for developing highly dynamic web applications. Among its features are: layout system, list and collection views, basic object model, custom UI behavior and a radio messaging system.

**UnderscoreJS** <sup>28</sup>This library provides functional programming functions to JavaScript, such as: map; reduce; and invoke. It aims to improve the productivity of the development of JavaScript applications.

---

<sup>24</sup><http://getbootstrap.com/>

<sup>25</sup><https://jquery.com/>

<sup>26</sup><http://backbonejs.org/>

<sup>27</sup><http://marionettejs.com/>

<sup>28</sup><http://underscorejs.org/>

**Leaflet** <sup>29</sup>This library has the ability to integrate maps into web pages. It has been built using JavaScript and supports mobile-friendly maps view. Also, it has a very small fingerprint in size and it has a simple API.

**TurfJS** <sup>30</sup>This library provide an implementation for geospatial analysis functions in JavaScript. It supports browser and NodeJS environments and it aims to be fast in its calculations. It provides many geospatial operations and it uses GeoJSON as geospatial data formal.

**alaSQL** <sup>31</sup>This library is a SQL database (SQL-99) for JavaScript. It can run in browsers and NodeJS environment. Its main focus is to be very efficient and fast in executing queries. It can load and store data using a variety of formats including: CSV, JSON, TAB, IndexedDB, LocalStorage and SQLite.

### 5.4.3 Limitations

Indoor Location Management Tool is at a early stage of development. It aims to be a complete suite for managing Indoor Location databases so the application is very complex and challenging. The proposed work includes the basic implementation to be able to interact with the WPS 2.0 server in order to test the effectiveness of the WPS 2.0 standard for machine learning purposes.

In the next release, the query builder will be improved. The user will be able to specify for each property of an element the condition that should be satisfied in order to select an element. The user may specify conditions like: "contains"; "greater then"; "equals" and "less then"

---

<sup>29</sup><http://leafletjs.com/>

<sup>30</sup><http://turfjs.org/>

<sup>31</sup><https://github.com/agershun/alasql>



#### **5.4.4 Source code**

Git is used for managing this project. Bitbucket is used as remote repository. The repository that holds the code of this project is currently private, it will be published in the recent future. Copies of the code can be given under request.

## 6 Discussion

The proposed hypothesis of the thesis was not easy to validate. Proving performance improvements in modern Software environments is challenging. The resulting performance depends on plenty of factors including the machine on which the Software is running. Even more challenging is the prove of application responsiveness.

Therefore, the experiments are focused on the status of the client over time: blocked or idle. In blocked stratus, the client cannot perform any operation since it is waiting for the response of the server. While in idle, the client can perform other actions, such as request resources from the web, respond to user input and update the GUI.

The gathered data come from two simulations, one (Large dataset) with 15000 elements and the other (Small dataset) with 3000 elements. The data is sent to the wps-classificator web service using the async-wps library. The WPS payload that included the samples were sent using a XML file. The results come from the last of 20 executions for each simulation. This is due the optimization the server performs during the execution of repeated operations.

Table 25 shows the results of the experiments. The table includes the amount of time the client remains in Blocked and Idle status for each simulation. Also, the total elapsed time is presented. Given this results, the asynchronous execution of the simulations led to significant availability of the client versus the synchronous execution. Now the client is in Idle status for 95.3% of the time for the Large dataset and 95.5% of the time for the Small dataset.

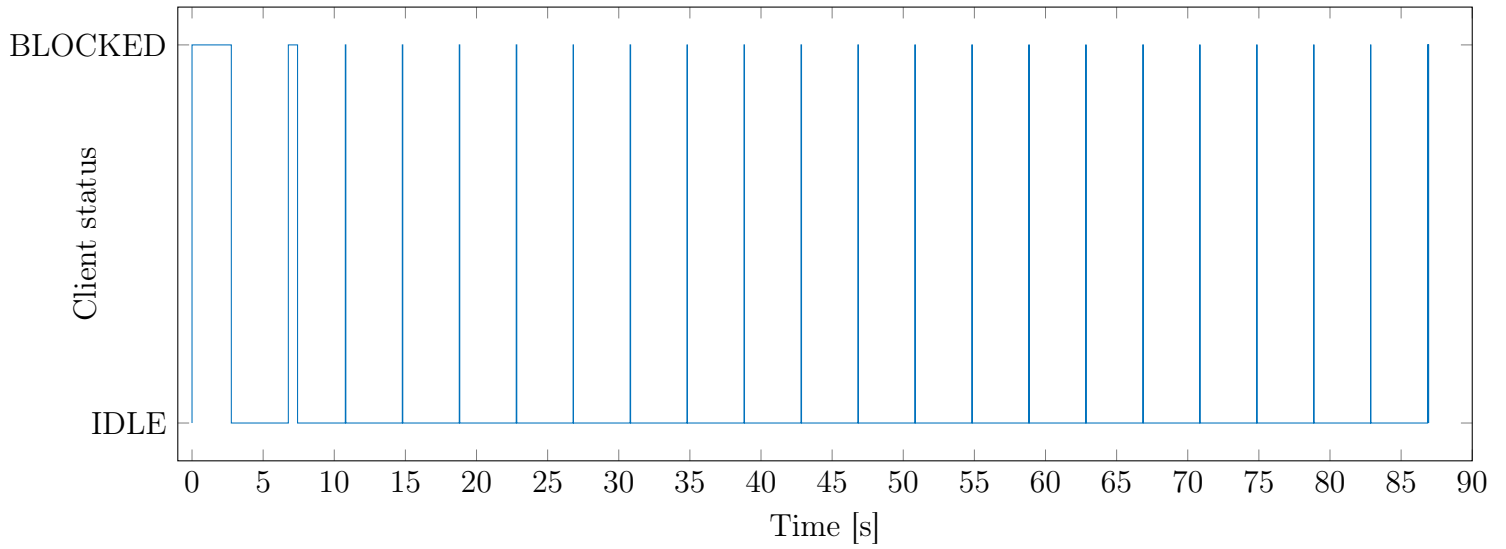
Figure 9 shows the client availability over time during the simulation with the Large dataset. Figure 10 shows the same information referred to the simulation using the Small dataset. These graphical representation a presented to better understand the results.

The experiments using the WPS 2.0 asynchronous feature led to another consideration. As described in Section 3.3, the polling technique used for getting the status of a scheduled execution is an issue of the WPS standard. In the current version of WPS, the

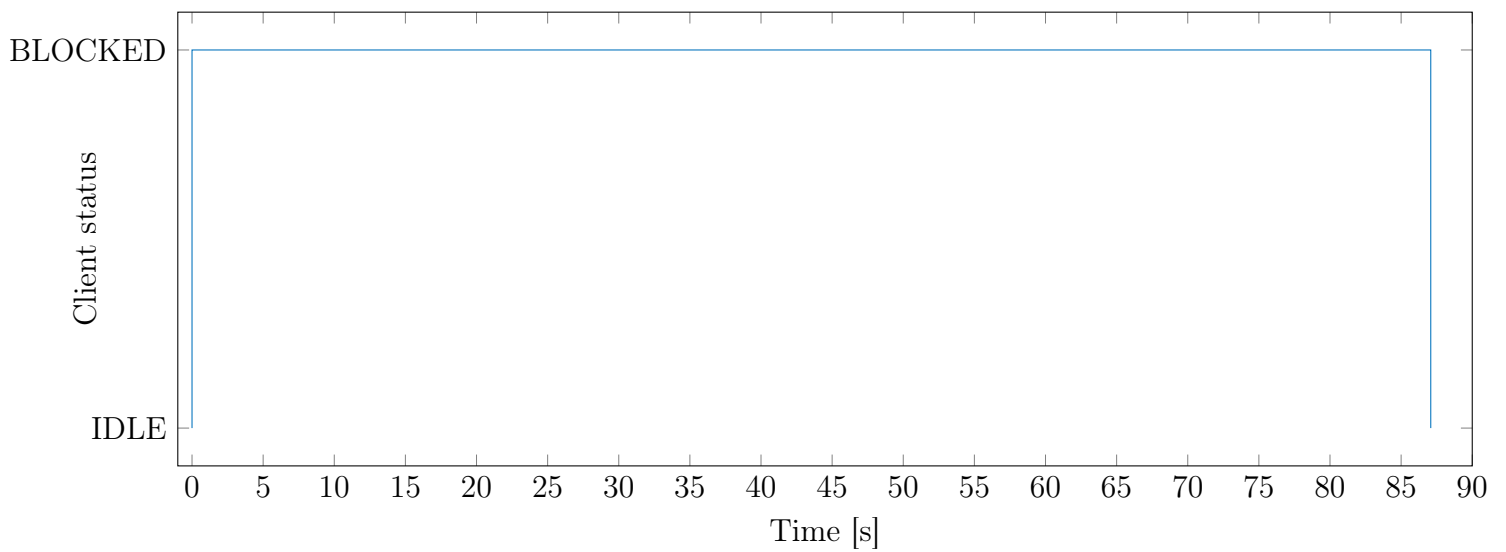
client must get the status of a scheduled process execution periodically (polling). The selected interval between GetStatus requests may influence the performance significantly. Although the WPS server may specify the preferred polling interval, it may not be the best one. During the experiments, this problem came out and a further study is presented in Figure 11. The figure shows how the polling interval affects the completion time of the process execution. Given the presented results, the polling technique is clearly not the best one for getting the status of a WPS process, since a wrong polling interval may halve the performance of the entire system.

	Large dataset			Small dataset		
	Blocked (s)	Idle (s)	Total (s)	Blocked (s)	Idle (s)	Total (s)
Asynchronous	4.04	82.88	86.93	0.37	7.98	8.35
Synchronous	87.08	0	87.08	8.12	0	8.12

Table 25: This table shows the status of the client during an asynchronous and a synchronous request for each simulation. The large dataset involves 15000 elements, while the Small dataset only 3000 elements. During the Blocked status, the client cannot perform any action, since it is waiting for the response of the server. During the Idle status, the client is available for doing other tasks, like updating the GUI or processing other data. The time is expressed in seconds

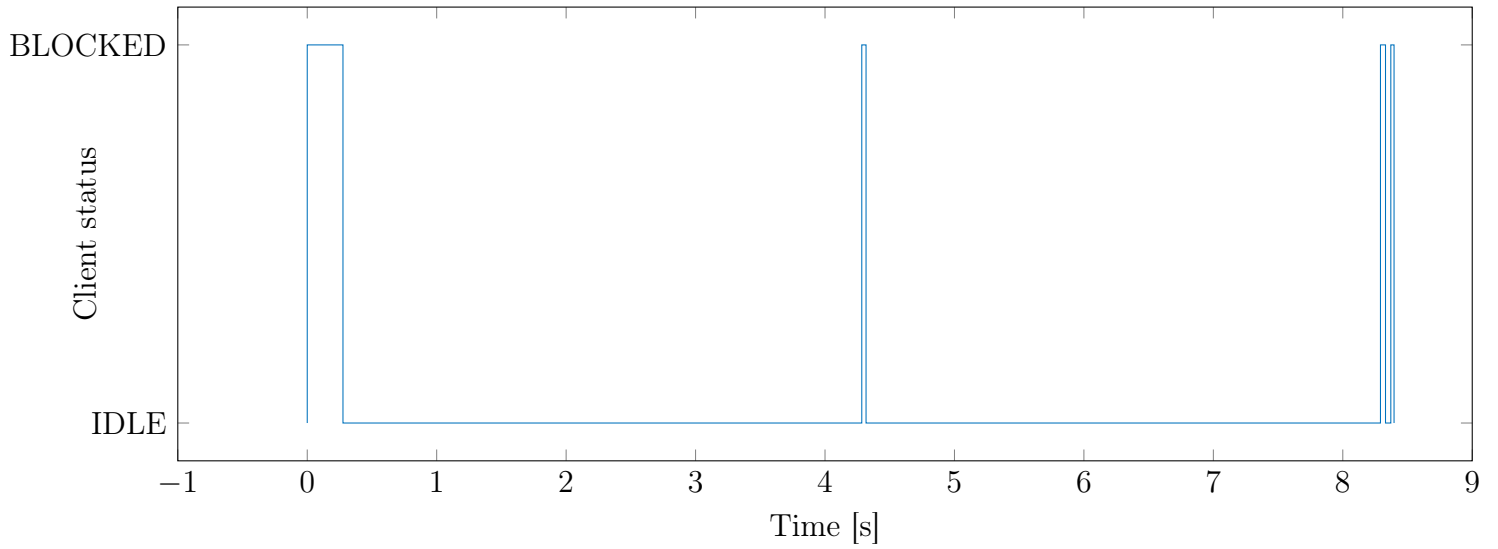


(a) Client status chart for an asynchronous request using the Large dataset

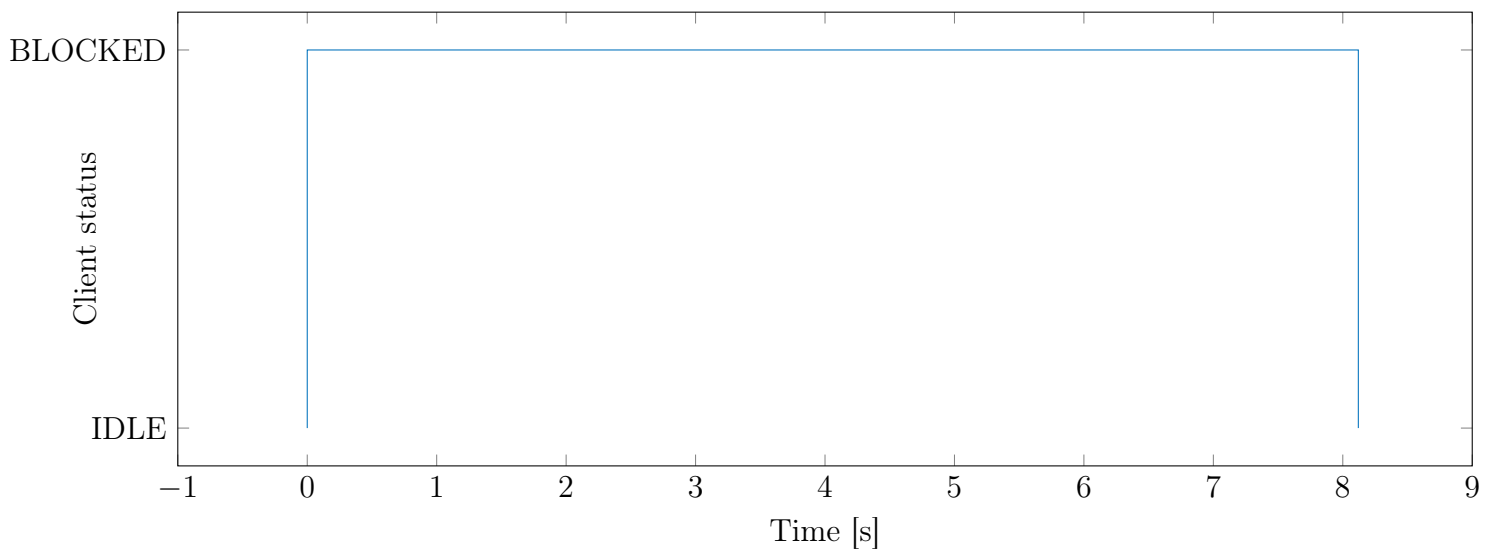


(b) Client status chart for a synchronous request using the Large dataset

Figure 9: This chart compares the status of a WPS client during asynchronous (a) and synchronous (b) requests for the Large dataset. The simulation is particularly heavy since it involves 15000 elements

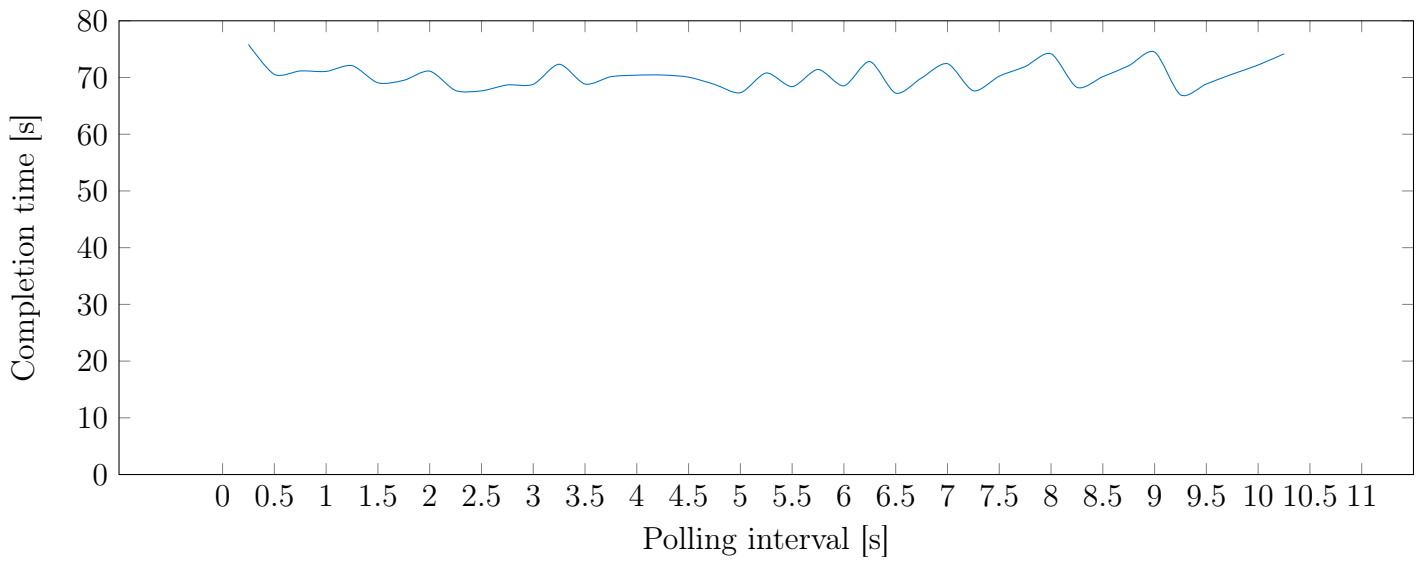


(a) Client status chart for an asynchronous request using the Small dataset

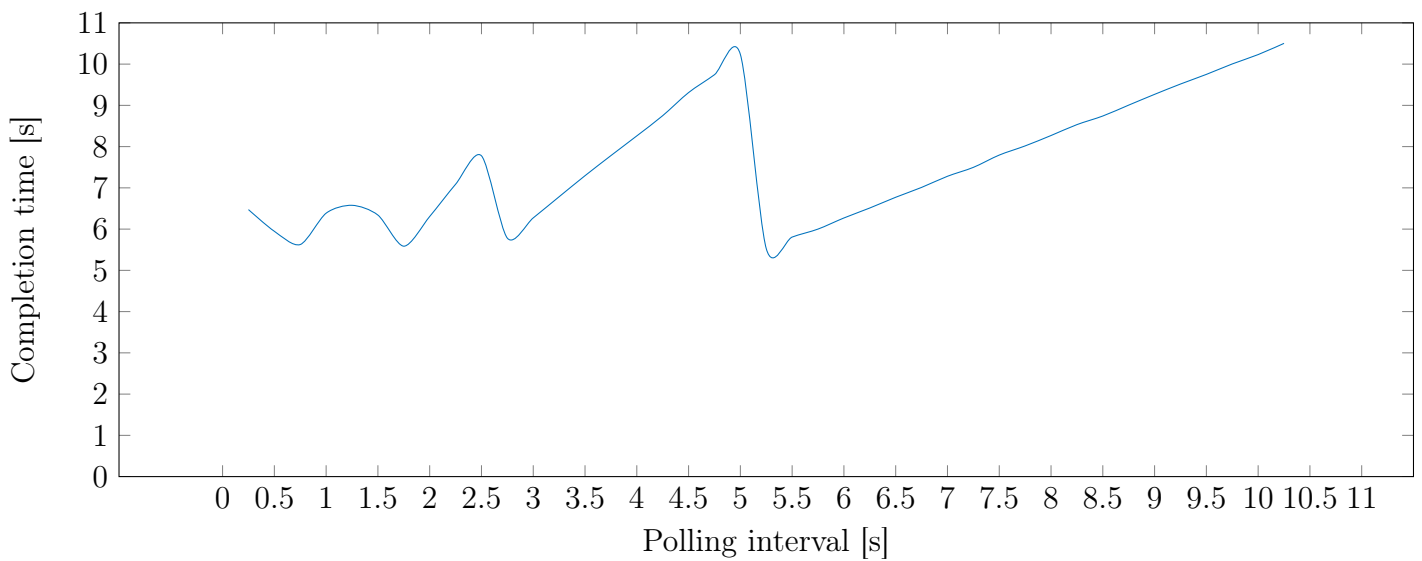


(b) Client status chart for a synchronous request using the Small dataset

Figure 10: This chart compares the status of a WPS client during asynchronous (a) and synchronous (b) requests for the Small dataset. The simulation is particularly heavy since it involves 15000 elements



(a) Completion time versus polling interval for the experiment Large dataset. Average: 70.36 seconds. Standard deviation: 1.69



(b) Completion time versus polling interval for the experiment Small dataset. Average: 7.62 seconds. Standard deviation: 1.29

Figure 11: These charts show the evolution of the completing time for the proposed simulations compared with the interval between status polling. (a) shows the results for the Large dataset, while (b) shows the results using the Small dataset. The polling interval (in seconds) is the time the client waits before requesting the status of the execution. The selected polling interval affects significantly the total execution time of the experiments depending on the duration of the experiment (in seconds). Longer experiments (execution time) seems not to be heavily affected

## 7 Conclusion

The main contribution of this thesis is the development of a JavaScript library for accessing WPS 2.0 web services in an asynchronous way. Specially in the case of intensive computing tasks, this feature improves the client responsiveness, reduces network resources usage and allows the implementation of high-performance server infrastructures.

Version 2.0 of WPS introduced the ability to execute processes in asynchronous mode along with standard synchronous. This feature makes WPS suitable for processing large quantity of data or for using time-consuming algorithms. This is specially useful in a data analysis scenario in which there is a big amount of data that is needed for statistical and big-data analysis.

Indoor Location is suitable example of performing expensive machine learning algorithm on large datasets.

In such a context, giving a capable web service, it is demonstrated that for expensive calculations involving a large quantity of data, the asynchronous feature of WPS 2.0 and `async-wps` led to huge availability gain in the client side (around 95% in the performed experiments). Now the client is not blocked anymore for long time waiting for the server response. During the server calculations, the client is now able to perform other tasks and provide early failure detection for the scheduled task. Also, the regular query of the process status improves the friendliness of the user interface and give real time updates to the user.

The proposed work consists of three Software programs. The first one is `async-wps` which is a client library for accessing WPS 2.0 services. According to the accomplished requirements (Table 22), the library features:

- Support for WPS 2.0 operations: `GetCapabilities`, `DescribeProcess`, `Execute`, `GetStatus` and `GetResult`.
- Compatibility with NodeJS and browsers (e.g. Chrome and Firefox).

- Support for ComplexData type as input for the WPS server.
- Open source license.
- Simple JavaScript API that supports asynchronous execution using Promise API and automatic transformation between JavaScript objects and XML documents.

The second Software is a WPS 2.0 server for performing kNN classification. According to the requirements specified in Table 23, the web server is able to:

- Provide a web API using a simple and unique entry point.
- Support for all the operations of WPS 2.0 standard.
- Support for kNN classification. The input data can be specified either using HTTP POST body or XML files. This operation can be done asynchronously as well as synchronously.

The last Software is a web application called Indoor Location Management Tool. It is still a development phase, yet the requirements for the scope of this thesis are accomplished (Table 24). In short:

- Support for tabular and map visualization of the Indoor Location database. In both cases, it is possible to filter the database entries using form controls and geospatial operations. The type of filter depends on the type of the selected property (geospatial or alphanumeric).
- Support for the creation of clusters of database entries, called experiments. This option enables the batch execution of statistical analysis and allows a post-execution comparison.
- Open source license.



In conclusion, WPS 2.0 standard represent a solid communication protocol for process-intensive web services. The use case for this thesis implemented a simple kNN classification algorithm, but the same idea can be extended to more complex algorithm and more complex input datasets. A further revision of WPS for improving the GetStatus workflow would improve the performance of the standard.

## 8 Future work

The proposed work in this thesis is meant to be a starting point for future studies.

Concerning `async-wps`, the library will be updated to cover 100% of the WPS 2.0 standard and other OWS data models. Once this work will be completed, it will be proposed to OGC as the official client for accessing WPS 2.0 web services for JavaScript.

The `wps-classificator` web service will be published on a public accessible host and made available to everyone. Also, more classification options and algorithm will be introduced. Since the project is open source, the community can improve and upgrade the existing code.

The Indoor Location Management Tool will be updated according to future usability tests. This tool can be very useful for data scientist that are studying Indoor Location databases.

Finally, a comprehensive test of a WPS 2.0 interface versus an ad-hoc one will be performed. This test is very interesting because it will quantify the overhead of the WPS standard compared to a raw solution. A specially interesting study may include the usage of modern technologies like WebSockets or HTTP2 as a substitute of the polling technique used for getting the status of a scheduled process.

## References

- [1] ISO 8601. Data elements and interchange formats – information interchange – representation of dates and times. Technical report, ISO, [http://www.iso.org/iso/catalogue\\_detail?csnumber=40874](http://www.iso.org/iso/catalogue_detail?csnumber=40874), 2004.
- [2] Anthony M. Castronova, Jonathan L. Goodall, and Mostafa M. Elag. Models as web services using the Open Geospatial Consortium (OGC) Web Processing Service (WPS) standard. *Environmental Modelling Software*, 41:72–83, mar 2013.
- [3] Yi-Chao Chen, Ji-Rung Chiang, Hao-hua Chu, Polly Huang, and Arvin Wen Tsui. Sensor-assisted wi-fi indoor location system for adapting to environmental dynamics. In *Proceedings of the 8th ACM international symposium on Modeling, analysis and simulation of wireless and mobile systems - MSWiM '05*, page 118, New York, New York, USA, 2005. ACM Press.
- [4] Yin Chen, Dimitrios Lymberopoulos, Jie Liu, and Bodhi Priyantha. Indoor Localization Using FM Signals. *Transactions on Mobile Computing (TMC)*, 2013.
- [5] Zeqiang Chen, Nengcheng Chen, Chao Yang, and Liping Di. Cloud Computing Enabled Web Processing Service for Earth Observation Data Processing. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 5(6):1637–1649, dec 2012.
- [6] Jaewoo Chung, Matt Donahoe, Chris Schmandt, Ig-Jae Kim, Pedram Razavai, and Micaela Wiseman. Indoor location sensing using geo-magnetism. In *Proceedings of the 9th international conference on Mobile systems, applications, and services - MobiSys '11*, page 141, New York, New York, USA, 2011. ACM Press.
- [7] T. Cover and P. Hart. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13(1):21–27, jan 1967.
- [8] Karan Dhiman and Benson Quach. Google’s go and dart: Parallelism and structured web development for better analytics and applications. In *Proceedings of the 2012*

- Conference of the Center for Advanced Studies on Collaborative Research, CASCON '12*, pages 253–254, Riverton, NJ, USA, 2012. IBM Corp.
- [9] Constantinos Dovrolis, Brad Thayer, and Parameswaran Ramanathan. HIP. *ACM SIGOPS Operating Systems Review*, 35(4):50–60, oct 2001.
- [10] Gérald Fenoy, Nicolas Bozon, and Venkatesh Raghavan. ZOO-Project: the open WPS platform. *Applied Geomatics*, 5(1):19–24, mar 2013.
- [11] E. Fortuna, O. Anderson, L. Ceze, and S. Eggers. A limit study of javascript parallelism. In *Workload Characterization (IISWC), 2010 IEEE International Symposium on*, pages 1–10, Dec 2010.
- [12] Degree Framework. Website. <http://www.deegree.org/>. Accessed: 2015-12-20.
- [13] Gregory Giuliani, Stefano Nativi, Anthony Lehmann, and Nicolas Ray. {WPS} mediation: An approach to process geospatial data on different computing backends. *Computers Geosciences*, 47:20 – 33, 2012. Towards a Geoprocessing Web.
- [14] Carlos Granell, Laura Díaz, Alain Tamayo, and Joaquín Huerta. Assessment of OGC Web Processing Services for REST principles. (arXiv:1202.0723), feb 2012. Comments: 25 pages, 3 Figures, 2 Tables. Paper accepted to International Journal of Data Mining, Modelling and Management.
- [15] E. Hazzard. Openlayers 2.10 beginner’s guide. Technical report, 2011.
- [16] IETF. Tags for identifying languages. Technical report, Internet Engineering Task Force, <https://www.ietf.org/rfc/rfc4646.txt>, 2006.
- [17] Internet Engineering Task Force (IETF). Hypertext transfer protocol version 2 (http/2). Technical report, IETF, <http://tools.ietf.org/pdf/rfc7540.pdf>, 2015.
- [18] Internet Engineering Task Force (IETF). Known issues and best practices for the use of long polling and streaming in bidirectional http. Technical report, IETF, <http://www.hjp.at/doc/rfc/rfc6202.html>, 2015.

- [19] Ecma International. Ecmascript® 2015 language specification. Technical report, Ecma International, <http://www.ecma-international.org/ecma-262/6.0/index.html>, 2015.
- [20] K. Kaemarungsi. Distribution of WLAN Received Signal Strength Indication for Indoor Location Determination. In *2006 1st International Symposium on Wireless Pervasive Computing*, pages 1–6. IEEE.
- [21] E. Kazakov, A. Terekhov, E. Kapralov, and E. Panidi. WPS-based technology for client-side remote sensing data processing. *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, XL-7/W3:643–649, apr 2015.
- [22] Dimitrios Lymberopoulos, Jie Liu, Xue Yang, Romit Roy Choudhury, Vlado Handziski, and Souvik Sen. A realistic evaluation and comparison of indoor location technologies. In *Proceedings of the 14th International Conference on Information Processing in Sensor Networks - IPSN '15*, pages 178–189, New York, New York, USA, 2015. ACM Press.
- [23] I. Marin-Garcia, P. Chavez-Burbano, A Munoz-Arcentles, V Calero-Bravo, and R Perez-Jimenez. Indoor location technique based on visible light communications and ultrasound emitters. In *2015 IEEE International Conference on Consumer Electronics (ICCE)*, pages 297–298. IEEE, jan 2015.
- [24] Danut Mihon, Teodor Stefanut, Victor Bacu, Cosmin Nandra, and Dorian Gorgan. The geo-spatial service integration in educational domains by WPS compliant standard means. In *2014 IEEE 10th International Conference on Intelligent Computer Communication and Processing (ICCP)*, pages 383–389. IEEE, sep 2014.
- [25] Lionel M. Ni, Yunhao Liu, Yiu Cho Lau, and Abhishek P. Patil. LANDMARC: Indoor Location Sensing Using Active RFID. *Wireless Networks*, 10(6):701–710, nov 2004.
- [26] OGC. Wps 1.0 interface standard. Technical report, Open Geospatial Consortium, [http://portal.opengeospatial.org/files/?artifact\\_id=24151](http://portal.opengeospatial.org/files/?artifact_id=24151), 2007.

- [27] OGC. Ogc web services common standard. Technical report, Open Geospatial Consortium, [http://portal.opengeospatial.org/files/?artifact\\_id=38867](http://portal.opengeospatial.org/files/?artifact_id=38867), 2010.
- [28] OGC. Wps 2.0 interface standard. Technical report, Open Geospatial Consortium, <http://docs.opengeospatial.org/is/14-065/14-065.html>, 2015.
- [29] Open Geospatial Consortium (OGC). Website. <http://www.opengeospatial.org/ogc>. Accessed: 2015-12-20.
- [30] Victoria Pimentel and Bradford G. Nickerson. Communicating and Displaying Real-Time Data with WebSocket. *IEEE Internet Computing*, 16(4):45–53, jul 2012.
- [31] Nissanka B. Priyantha, Anit Chakraborty, and Hari Balakrishnan. The Cricket location-support system. In *Proceedings of the 6th annual international conference on Mobile computing and networking - MobiCom '00*, pages 32–43, New York, New York, USA, 2000. ACM Press.
- [32] Mohamed Er Rida, Fuqiang Liu, Yassine Jadi, Amgad Ali Abdullah Algawhari, and Ahmed Askourih. Indoor Location Position Based on Bluetooth Signal Strength. In *2015 2nd International Conference on Information Science and Control Engineering*, pages 769–773. IEEE, apr 2015.
- [33] Beate Stollberg and Alexander Zipf. OGC Web Processing Service Interface for Web Service Orchestration Aggregating Geo-processing Services in a Bomb Threat Scenario. In *Web and Wireless Geographical Information Systems*, pages 239–251. Springer Berlin Heidelberg, Berlin, Heidelberg.
- [34] Joaquín Torres-Sospedra, Raúl Montoliu, Sergio Trilles, Óscar Belmonte, and Joaquín Huerta. Comprehensive analysis of distance and similarity measures for Wi-Fi fingerprinting indoor positioning systems. *Expert Systems with Applications*, 42(23):9263–9278, dec 2015.
- [35] Joaquin Torres-Sospedra, David Rambla, Raul Montoliu, Oscar Belmonte, and Joaquin Huerta. UJIIndoorLoc-Mag: A new database for magnetic field-based lo-

- calization problems. In *2015 International Conference on Indoor Positioning and Indoor Navigation (IPIN)*, pages 1–10. IEEE, oct 2015.
- [36] Roy Want, Andy Hopper, Veronica Falcão, and Jonathan Gibbons. The active badge location system. *ACM Transactions on Information Systems*, 10(1):91–102, jan 1992.
- [37] WPSint. Website. <http://wpsint.stage.tigris.org/>. Accessed: 2015-12-20.
- [38] Yan Yan Zhao, Xue Feng Liu, Jian Hua Mao, Xiao Ling Yang, and Hao Ran Wang. 52 North WPS and its Application in Fire Emergency Response. *Advanced Materials Research*, 760-762:1748–1752, sep 2013.
- [39] Chao Zhou, Houyao Xie, and Jiaoyang Shi. Wi-fi indoor location technology based on k-means algorithm. In Zhenji Zhang, Zuojun Max Shen, Juliang Zhang, and Runtong Zhang, editors, *LISS 2014*, pages 765–770. Springer Berlin Heidelberg, 2015.