UNIVERSITY OF DEUSTO

GeoWorldSim: A time-asynchronous, distributed and intelligent environment based Geosimulation platform

Tesis doctoral presentada por

ANDER PIJOAN

dentro del Programa de Doctorado en

INGENIERÍA PARA LA SOCIEDAD DE LA INFORMACIÓN Y DESARROLLO SOSTENIBLE

dirigida por

DR. CRUZ E. BORGES HERNÁNDEZ y DRA. AINHOA ALONSO VICARIO

UNIVERSITY OF DEUSTO

GeoWorldSim: A time-asynchronous, distributed and intelligent environment based Geosimulation platform

Tesis doctoral presentada por

ANDER PIJOAN

dentro del Programa de Doctorado en

INGENIERÍA PARA LA SOCIEDAD DE LA INFORMACIÓN Y DESARROLLO SOSTENIBLE

dirigida por

DR. CRUZ E. BORGES HERNÁNDEZ y DRA. AINHOA ALONSO VICARIO

El doctorando | El Co-director | La Co-directora

To all of you who had an impact in my life. Find yourself here:

A E D P C P G R Z E G O R Z J Q U E R Y
N N O H H U B E R T N N R G D J A N G O
A S I E R A N G E L A L V A R O J A D E
M A R C I N I E T O N Y A N O D E J S G
A M E T S J R E U I D I E G O E L U A W
I I G O T O A I N H O A S P E P E L N S
A K C R I S T I N A I D O A M A L I N A
L E R P A U L A S N M A T M P P A N L I
E L U A N A J O S E I G N A C I O I O P
N I Z T P T U N A I K E W M E R G O R S
I R I P A B L O D R E T U A L E P A E U
D A G G I D E A E C L I M I I R P E N M
A I N O D A N I R A E X A M A L I N A S
R A A R E V A M W R X Z R W Z P E R A I
L X C K X I V A N L U A I T O R D I M G
I Z I A N D O N I O R S J O S E A K A T
S L O R E A F O Y S I K E R R H D E S E
X J V A N G U L A R P 0 1 E D E U S T O

If you can't find yourself, try harder. I'm sure you are here somewhere.

Abstract

Imagine that the accessibility of the population to public infrastructures needs to be evaluated. A possible solution would be to calculate every distance and analyse which percentage of the population is in less than 300 meters. However, this solution does not take into account issues such as: the age distribution, functional diversity or people's preferences when transiting the city. It is in these cases when it is necessary to go a step further and integrate Geographic Information Systems with other perspectives such as Multi-Agent Systems which represent the particular characteristics of each individual and their decision making processes. This integration is known as Geosimulation and builds more accurate simulations to model the physical reality together with social, demographic and economic components.

Geosimulations aim at modelling systems at the scale of individuals and entity-level units of the built environment and provides a way to simulate big amounts of agents interacting in a virtual geographic environment and endowed with spatial cognitive capabilities (perception, navigation, reasoning). This dissertation presents a new Geosimulation platform design and implementation that allows analysing and simulating different urban infrastructures. The platform manages to put into practice the latest theories in Multi-Agent Systems along with the new techniques in cloud computing and asynchronism. The proposed design is evaluated for three case studies; ubiquitous IoT, sustainable transport policies and resilience of the power grid. The methodologies presented, provide

progress to their respective research areas by improving state-of-the-art techniques or designing new mechanisms.

Furthermore, by connecting these Geosimulations to the real world by sensors and actuators, the concept of mixed reality arises; simulations where changes in the real world are transferred to the virtual world through sensors and agents can influence the real world through actuators. Mixed realities allow developing distributed control systems, which not only take into account the physical reality and social preferences but also the state, where to deploy intelligent agents that provide services to citizens.

Acknowledgements

This research was partially funded by

- GREENTRAVELLING project (Era NET Transport 6/12/IN/2014/195 label) partially funded by Diputación Foral de Bizkaia through the Plan de Promoción de la Innovación (6/12/-IN/2014/195) and the Basque Government through the GAITEK program IG-2014/0000133)
- S-MILE project (Era NET Transport ENTIII Flagship 2015 label) funded by Basque Government through the HAZITEK program (ZL-2017/00081)
- Industrial Ph.D. grant given by the University of Deusto (2015-2018)

This dissertation uses icons and images provided by

- Icons made by www.flaticon.com free icon stock
- Icons made by DinosoftLabs for www.flaticon.com
- Icons made by Vectors Market for www.flaticon.com
- Icons made by Freepik for www.flaticon.com
- Icons made by Twitter for www.flaticon.com
- ESRI codebase

Contents

List of Figures

List of Tables

List of Listings

Acronyms

ABM Agent-Based Modelling.
ACL Agent Communication Language.
AI Artificial Intelligence.
API Application Programming Interface.

CA Cellular Automata.
CPU Central Processing Unit.
CSS Cascading Style Sheets.
CSV Comma Separated Value.

DOM Document Object Model.
DSO Distribution System Operator.

EDSS Environment Decision Support System.

FIPA Foundation for Intelligent Physical Agents.

GIS Geographic Information System.
GPS Global Positioning System.
GUI Graphical User Interface.

HTML Hypertext Markup Language.
HTTP Hypertext Transfer Protocol.

IoT Internet of Things.

JSON JavaScript Object Notation.
JSON-LD JavaScript Object Notation for Linked Data.
JVM Java Virtual Machine.

LSMAS Large Scale Multi-Agent System.
LSS Large Scale System.
LTLF Long Term Load Forecasting.

MAS Multi-Agent System.

SHP Shapefile format.
SLF Spatial Load Forecasting.
SQL Structured Query Language.
SSO Single Sign-On.
SVM Super Vector Machines.

XML eXtended Mark-up Language.
XSS Cross-Site Scripting.

CHAPTER

1

Introduction

Urban planning is a technical and political process concerned with the design and regulation of the urban space, economic functions, social impacts and activities within it. Their evaluation and planning is a difficult task currently under consideration by many research centres and commercial tools. It involves evaluating processes and networks for managing power, water, transportation, communications or waste networks among others. Urban planning concerns itself with both the improvement of the public welfare as well as effects on everyday social and economic activities. While there is widespread consensus on how each urban process should work by separate, when combining all, most major planning decisions involve trade-offs between confronted objectives and thus conflicts emerge. Good urban planning implies goal setting, data collection and analysis, forecasting, design, strategic thinking and public consultation.

To this end, Geographic information technologies have been a standard to map existing urban systems and to project the consequences of changes. It is an obligation then to start with a brief introduction of what these are and how they work.

1.1 Geographic Information Systems

Cartography is the art and science of making maps. From cave drawings to ancient maps, people have used maps as essential tools to help them define, explain, and navigate their way through the world. Maps are a product of human endeavour always created to sever a purpose. These display only a few selected features with processed information portrayed in highly symbolic styles according to a classification scheme and certain basic assumptions which are not always true or verifiable, but useful. Imagine a map where each ocean is coloured differently. It is not true that water has those colours, but that differentiation would allow knowing the bounds of each ocean. In these ways, all maps are estimations, generalisations and interpretations of true geographic conditions and, as such, may be subject to unwitting errors, misrepresentation, bias, or outright fraud. As the famous quote from George Box (Box, 1979) states, "All models are wrong, but some are useful". In this sense, maps have proven to be remarkably adaptable and useful through several millennium of human civilisation and are fundamentally important for modern society.

Geographic Information Systems (GISs) are software applications to deal with spatial information on a computer, that is, to create digital maps, manage elements in them and perform spatial analysis with such information. There is a generally accepted assertion stating that around 80 percent of all world information is geographically referenced (Hahmann et al., 2011).

GIS make it possible to integrate different kinds of geographic information, such as digital maps, aerial photographs, satellite images and Global Positioning System (GPS) data, along with associated tabular database information (e.g., attributes or characteristics about geographic features). This geographical component brings rich and pervasive information that enables a better understanding of the data, more efficient operation and more competitive to institutions. GIS help to perform statistical analysis or spatial queries, to explore *what-if* scenarios, and to create predictive models by execution operations such as:

- What exists at a given location?
- Where does something occur?

- Which is the distance or area between two elements?
- What has changed since a specific point in time?
- How to reach from point A to B?

Institutions and companies have generated during the last decades new insights, business models and knowledge by mapping location-specific data (known as GIS data) in fields such as natural resource management (Powar et al., 2018), infrastructure and transportation planning (Guerreiro et al., 2018; Sierpiński et al., 2014) or health and public safety (Rodrigues et al., 2015). GIS data has tremendous potential for facilitating decision-making process and for revealing the combined effect of incremental decisions. Indeed, many have praised GIS for their ability to provide *better information* - information that is faster, cheaper, more reliable, more readily available, and more understandable - which, in turn, might lead to *better decision-making* (O'Looney, 1997).

1.1.1 **GIS Data**

GIS improved traditional information data-sets extending the allowed data types and operations to more than text or numbers. Depending on the system, the new data types offered can be the ones described in Table 1.1. Using these types, data-sets can be extended to host important information that otherwise would not be possible. Table 1.2 shows a (non-geographical) table extended with geographical data containing the town hall location and boundary of a municipality. It is true that the location of the town hall could have been stored with two number typed columns for latitude and longitude but as soon as geometries become more complex (e.g. the case of polygons) there would not be a clean way to manage them using standard data types. In addition, GIS applications can store more than one geographical aspect associated to one element, something that paper maps are not very good at. A GIS could draw the municipality boundaries from Table 1.2 adding a colour scale based on population or on the calculated area of the boundary. So within a GIS, maps' appearance can be easily changed based on geographical and non-geographical data associated to map elements.

Table 1.1: GIS data types. (Source: Own research)

Type	Description
Point	Single coordinate for representing the location of an element.
Line	Ordered sequence of more than two points that defines linear features such as rivers and roads.
Polygon	A closed line delimiting an area or volume. It can contain holes (represented as inner polygons) within.
Multipart	Combination of any of the above three types.
Graph	Set of points joined by lines so that they build a connected network.

Table 1.2: Example of Points and Polygons data types provided by GIS for municipalities and town halls. (Source: Own research)

name *text*	**population** number	**townhall** *point*	**boundary** *geometry*
Balmaseda	7700	–3.2,43	–3.2,43 –3.2,43 –3.2,43 ...
Arrigorriaga	12 000	–2.9,43	–2.9,43 –2.9,43 –2.9,43 ...
Bilbao	350 000	–2.9,43	–2.9,43 –2.9,43 –2.9,43 ...

GIS data is often modelled in two different formats according to the level of detail and the information to be represented:

Raster data: A grid surface composed of regular cells with a value associated to each. A digital photograph is an example of a raster dataset where each pixel value corresponds to a particular colour. Only the boundaries of the grid are georeferenced and extracting the location of all inner cells is straightforward. In GIS, cell values may represent information that is continuous along the area such as elevation above sea level, chemical concentrations, rainfall, etc. Therefore, independently of the geometries to represent, a raster grid covering the same area and amount of cells will always have the same size. However, if zooming in too much on a raster image, it will start to appear *blocky*, as the cells will get bigger without being spread into sub-cells.

Vector data: Spatial objects such as points, lines and polygons stored as a series of x, y and z coordinates. Vector data stores all the coordinates that must be joined in a particular order to recreate a geometry. This means that geometries will not lose level of detail no matter how much zooming is applied to the map. In GIS, vector data is used to shape information with discrete boundaries, such as country borders, land parcels, or streets. However all these coordinates can be excessive depending on the level of detail and amount of geometries leading to unmanageable datasets.

Figure 1.1 shows the two types of data being viewed in a GIS application. Translation between the vector and raster data models can be done using a number of conventional methods, though none is without error.

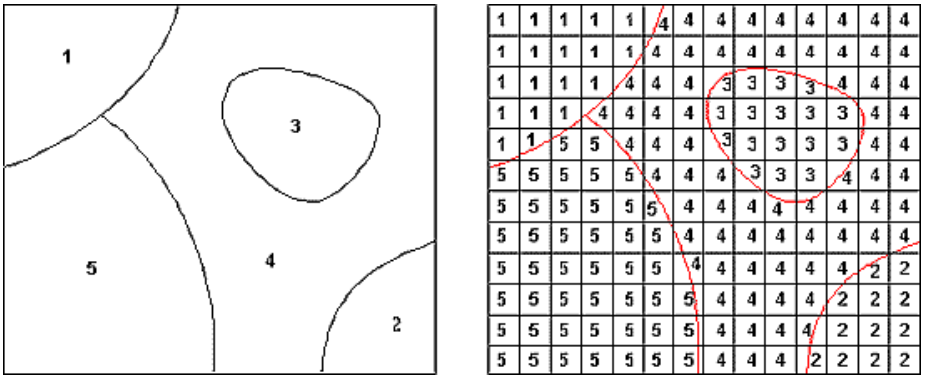


Figure 1.1: Map displayed as vector (left) and raster (right). Note that the red lines in the raster grid's are for better viewing of the geometrical data but are not stored in the file. (Source: Center for International Earth Science Information Network - Columbia University et al. (2014))

1.1.2 GIS Analyses

GIS processes automate tasks that were historically done manually. The main operations use the described data-types in Table 1.1 and create new maps by transforming or overlaying geometries on top of others. The main analyses GIS perform are:

Transformations: Geometric transformations are used to assign coordinates to map features in a GIS. Such transformations adjust the coordinates of a data layer so it can be correctly overlaid on another layer of the same area. For this purpose two approaches are used, absolute and relative.

Relative Position: Refers to the location of features in relation to a geographic coordinate system. A data layer or element is defined as correctly located and then the remaining features are adjusted with links representing from and to locations.

Absolute Position: The location in relation to the ground which requires transforming all the individual features. The advantage of the absolute methodology is that it does not propagate errors from one layer to the next.

Geometric unions and intersections: Combining features with unions (joining area A and B to create another one covering the area of both) and intersections (extracting the only part shared by two different areas A and B) is a widely used technique to create spatial information. For example, the intersection of a river area and highway line would determine the area where a bridge should be built.

Containment: Testing if a features is contained within another is a topological procedure which determines the spatial coincidence of points, lines and polygons. Geometries can be assigned the attributes of the polygons within they fall. For example, this function can be used to analyse an address and find out if it (point) is located within a certain zip code area (polygon) or the total miles of state highways within the boundaries of a certain county. Using vector data, the identification of points and lines contained within a polygon area is a geographical function comparing all the coordinates of both features. On the contrary, in raster data it is essentially an overlay operation, with the polygons in one data layer and the points and/or lines in a second data layer.

Buffer: Buffering is commonly used with proximity analysis to indicate the sphere of influence of a given element. Buffering involves creating a zone around a given point, line, or polygon of a specified distance. It is useful for further analysis

testing if features rely within the calculated area. For example, a 1000 m buffer could be generated around a school to then use overlay analysis to find out how many libraries are within 1000 m of that school.

Distances: Some distance analyses require the measurements to be constrained to a road, stream, or other linear network. This requires building a network graph where the nodes represent the physical points, edges represent the available connections (roads, pipes, etc) between those nodes and each edge contains the cost (distance, price, speed) of travelling using such.

Service area: Following with graph distances, a network service area is a region that encompasses all accessible points of a graph without exceeding a certain cost. It can be understood as the buffer operation from a point in a graph (that is, streets that are within a specified impedance). For instance, the 5 min service area for a point on a network includes all the streets that can be reached within 5 min from that point. Concentric service areas show how accessibility varies with impedance.

1.2 Limitations of traditional GIS analyses

When working with GIS, as knowledge about them grows, their limitations are rapidly glimpsed. It is not necessarily that GIS have shortcomings, but that their scope holds on to the representation and operation of geographical factors. All the analyses described in Section 1.1 work with a static capture of the data which does not change in time and with objective spatial factors that do not give rise to different interpretations. The following sections will provide some insides about the limitations that GIS technology have and why they do not provide complete results.

1.2.1 Representing relations

Some GIS features usually have spatial relations and constraints that can improve or even be critical for obtaining the results. There are many neighbouring and spatial calculations to detect these relations but sometimes they are not calculable with

just geometries comparing processes. In order to represent such relations between elements, GIS use the concept of Topology. It assumes that geographic feature relations are represented as nodes (points) and edges (connecting arcs between points). Thereby, nodes connected by an edge have a certain relation. However topology usage is limited to specific cases where geometries share some part (e.g. two polygons that touch themselves) or for binary element relations (e.g. being able to travel from one point to another).

While there are several standards for representing unitary geometries, there is no consensus on how to model spatial relations. Mappers create their custom topologies for their needs trying to fit all relations as pair of nodes connected by an edge. Figure 1.2 illustrates a graph with an intersection inside. Within the intersection, travellers arriving from B can not turn left to A and the ones arriving from D can not turn right to C. There is no standard way to represent said restrictions. A first approach to avoid computers sending a car ignoring those restrictions could be removing the intersecting point, but this point is necessary to model that there is an actual intersection (otherwise a computer could understand it as a crossing at two different levels). According to their necessities, GIS modellers have created their own custom ways of defining turn restrictions (Jiang et al., 2002) but implies creating complicated fictional road segments that require extra costly processing and blur the network. Furthermore, when relations imply many different elements (e.g. turn restrictions depending on the day, vehicle type and occupation) it will be even more difficult to define a schema to represent them.

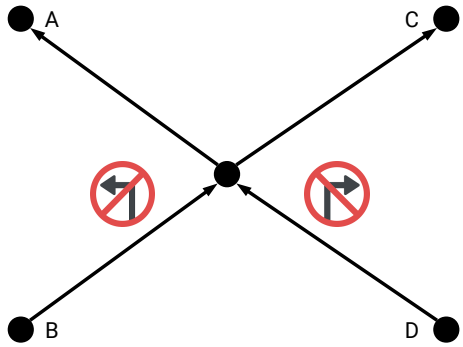


Figure 1.2: Example of a map where intersection turns must be restricted. (Source: Own research)

1.2.2 Centralised control

Many GIS techniques were formulated in a time in which urban areas had different shape than what they do today. In terms of scenarios, the conceptualisation was that cities were structurally monocentric, organised with a dominant centre and dispersing homogeneously with distance from the urban core and, thus, dispersion models were created based on these assumptions. This centralised approach does not adjust appropriately for urban dispersion models since many areas are decentralised in terms of activity, population or structure. Consequently, geo-processes need to escape from the traditional centralised view to bottom-up approaches in which cities have distributed rates of expansion that work in a decentralised way.

1.2.3 The world is not static

GIS work with a capture in time of current or past situation of a scenario. Geographic calculations may require big time and, knowing that the world changes during those experiments, for when results are obtained these may not be useful. To solve this, there are already some tools that add a time-stamp to the geographical features so that it is possible to know at what moment each one existed. However, going back and forth in time in these tools is simply turning elements off and on without really

evolving and modelling that transformation process for each feature. GIS have failed to representing the temporal and evolutionary component of a territory.

This issue is probably due to the fact that developers and scientists have primed the modelling of objects over processes, so GIS are good at representing the map but not the processes (the model) that give rise to the map. This makes GIS tools very precise but also very cold. They do now take into consideration the social or physical processes that transform the map layout which are often more scientifically relevant than the map itself. On the one hand, a simple example of the physical dynamism would be mountain erosion where material is redistributed across the landscape over time. On the other, social dynamism could be water supply network where citizens consuming water results in water flows to be in constant redistribution.

Moreover, within the Internet of Things (IoT) paradigm (Xia et al., 2012), technologies are focusing on creating processes that actively monitor the real situation of the world, process it and in return actuate on it. These active listeners must understand the process that is behind the signals they receive in order to give a coherent and useful response.

Having identified all these limitations, it is clear that extending GIS to accommodate non-geographic factors would result in losing their primary mission and going against the KISS[1] (Terano, 2008) principle. KISS states that most systems work best if they are kept simple rather than made complicated; therefore simplicity should be a key goal in design and unnecessary complexity should be avoided. So bounded expertise has lead GIS to perfectly couple with other domain specific software in a mixed-methods relationship to improve environmental, economic or health care analyses. The question at this point was how to go a step further and integrate subjective aspects that each person understands in different ways. This is where MASs came to the rescue.

[1] Keep it simple stupid!

1.3 **Multi-Agent Systems**

Problems involving many interacting features are large in size, dynamic and stochastic by nature. Such Large Scale Systems (LSSs) are intricately multifarious, with multiple objectives that can lead to conflicts among the multiple decision makers present in the problem. A system can be considered as a LSS if one (or both) of the following perspectives holds:

- It can be decomposed into a number of interconnected subsystems, either for practical reasons (design) or because computation needs to be distributed (performance);
- Its high dimensionality leads to a combinatorial explosion in its space of possible behaviours, so that the usual methods for modelling analysing, controlling or designing cannot find a solution in a reasonable amount of time. As a result, these systems require the control of the data and/or computation to be decentralised over the subsystems.

Over the last decades, many universal approaches have been widely used for modelling stochastic processes, such as Monte Carlo techniques (Rubinstein and Kroese, 2016), Markov Chain models (Keilson, 2012) or Queuing Theory methods (Allen, 2014). Creating software representations of these problems allows having risk-free ways to explore the performance of said processes to extract knowledge before making real-world changes. In this sense, MAS have been an active area of research in the last three decades (Wooldridge and Jennings, 1995). What started as isolated systems for simple problem-solving have turned into full-scale simulation platforms that are able to integrate several state of the art technologies. MAS are computer systems composed by agents capable of interacting with each other. MAS aim at capturing collective behaviour of human decision-makers within the environment that supports their activities. In particular, MAS are a natural approach for modelling and implementing LSS because they rely on decentralised control by means of agents. Each intelligent agent comprises an independent program that emulates a behaviour and can move within the environment in order to fulfil its

objectives. Agents can communicate, negotiate, and learn; they can perceive their environment, process perceptions and respond or act in a rational manner as would the entity they represent. According to Macal and North (2010), MAS are composed of the following:

1. A set of agents, their attributes and behaviours.
2. A set of agent relationships and methods of interaction: An underlying topology of connections defines how and with whom agents interact.
3. The agent's environment where they live in and interact with it.

Macal and North (2010) also state that Agent is the definition applied to an entity that can be simulated by means of a computer. They are independent systems that try to fulfil a set of goals and emulate a certain behaviour. All these agents are situated in an environment they can sense and act upon, and are executed in parallel either competing with each other or cooperating to achieve a common goal. In this way, it is possible to create networks of agents working together to give response to complex problems that may need a collective result beyond what can be achieved with the individual knowledge of each agent.

The classic example of this is calculating the travel time of a large number of workers who leave at the same time to return from their working places to their homes by car. Calculating the time independently for each traveller would give a more than acceptable result. However, a large number of vehicles always involve delays and heavy traffic. Being able to simulate all the vehicles travelling at the same time and competing with each other to be the first vehicle to leave, reproduces traffic delays in a much more reliable way and consequently the journey times are closer to reality.

However, MAS simulations are of high dimension (lots of interacting features), large in size and often stochastic by nature. Bottlenecks in an Large Scale Multi-Agent Systems (LSMASs) are usually related to the size of the system in terms of the number of agents, amount of messages exchanged, computational needs and the amount of data in the system. Indeed, regardless of the application domain, each additional agent requires some computational resources. Therefore, the MAS should be able to

accept new agents without compromising its functioning. To limit the exponential need for computational resources, simulation models target an accuracy and scope, being classified as:

Micro-simulation models: Describe the world with a high level of detail and distinguish separate elements that live in it. Though the use of this high level of detail entails very precise analysis, it must be limited to a small area or intersection due to its complexity.

Meso-simulation models: Describe the world with an intermediate level of detail, for example by distinguishing separate elements in the traffic flow but not taking into account the interactions between them. They are less precise and can be applied to cover larger areas than those of the micro-simulation.

Macro-simulation models: Describe the world with a high level of aggregation, as a uniform flow. Macroscopic simulations have been traditionally developed to model large-scale areas or systems.

1.3.1 **The agent environment**

It has already been said that, agents in a MAS run and coexist in a so called agent environment as described by Weyns et al. (2005). Within MAS technologies, the agent environment concept has been recognised as an independent and living element which is essential to model dynamic real world problems. More than ten years ago, Weyns et al. (2005) already defined the scope and differences between the agent environment and agents that inhabit it. The real world environment is inherently dynamic as it changes beyond the humans' control. This dynamism is the main lack in traditional GIS and must be modelled explicitly as part of the simulated environment by implementing processes that can change its own state.

The agent environment is now broadly recognised as a first class abstraction for building a MAS, especially because it mediates interactions between agents and their access to resources. In an LSMAS, the number of interactions and resources could be very large, hence the design of the agent environment is even more crucial because

it directly impacts scaling issues and plays an important role in managing potential bottlenecks. Their foremost characteristics are:

Scalable Structure: refers to distributing the computation and state of the agent environment. The agent environment may have different structures: multi-level or hierarchical, multi-stage or dynamic. For example, the agent environment may be structured in segments, each representing a local view on the physical environment; segments may be connected via a Peer to Peer (P2P) network.

Access to Resources: Typically, LSMAS are composed of heterogeneous agents deployed in an agent environment, which defines laws that regulate access to resources. At a large scale, monitoring, trust, and security aspects are to be carefully designed so that the cost induced by managing access to resources does not become a bottleneck.

Scalable Communication: When coordination among thousands of entities is required, the agent environment should provide means for communication between agents that do not involve any central point of access or control.

Interaction model: to achieve scalable agent environments, it is important to provide agents with efficient means for perceptions, actions, and interactions. Central here are suitable abstractions, e.g., the agent environment should offer high-level primitives to agents for perception, coordination etc., that support efficient processing by the agents.

Regarding its implementation, there is plenty of literature describing the philosophy behind agent environments and how these should be structured. Weyns et al. (2007a) define a suitable software architecture that highlights the core functionalities that should be present in every agent environment such as:

- A deployment context that comprises sensors and actuators with which the MAS interacts.
- An abstraction level that bridges the conceptual gap between agent abstraction and low-level details.
- An interaction-mediation level that regulates the access to shared resources and mediates the interaction between agents.

Steiner et al. (2006) establish a design and implementation of an environment as a first-class entity for a geographically based simulation system. Finally, Valckenaers et al. (2007) establish that MAS applications can be heavily influenced by the structure of the environment in which their agents operate. An appropriate environment definition can help and bring together real world scenarios and applications by addressing key issues as time management, sensor processing, augmented reality and virtual actuators.

1.4 Joining GIS and MAS

Evaluating urban developments requires building more accurate maps that bring the physical reality together with social, demographic and economic components. Urban scenarios are the result of interactions between actors, over time, with feedback that changes the scenario in a continuous loop. In this sense, twenty years ago the first voices emerged stating that joining GIS and MAS would be beneficial for improving spatial sciences (Batty, 1999). It did not take long until the so called Geosimulations (Benenson et al., 2004) paradigm was born to improve the quality of the urban analyses that were done back then. The essence of the Geosimulations is to carry out a series of computational experiments of the real system and to generate output data that characterises the subject system. These can be distinguished from other simulation methodologies by its explicit attention to space and geography (hence the *geo*). Geosimulations allow to check, monitor and evaluate the behaviour of the real systems under different realistic conditions in an artificial computer based world.

However, the use of Geosimulations emerged small and not equitably for all research areas. Table 1.3 summarises how much effort and operational software has been carried out for building simulations by scientific area. More precisely, while GIS are transversal in all urban planning processes, surprisingly the use of MAS differs depending on the research area (Ouyang, 2014) and is even lower when simulating human behaviour regarding their interaction and resilience to urban systems.

Table 1.3: Geosimulation application domains in urban development. (Source: Own research)

Research area	Use of Geosimulations
Urban growth	Big (Chen, 2012)
Transportation modelling	Big (Bazzan and Klügl, 2014)
Power networks evaluation	Medium (Sujil et al., 2018)
Water networks evaluation	Medium (Teodosiu et al., 2009)
Waste management evaluation	Low (Karadimas et al., 2006)
IoT Communications evaluation	Low (Karnouskos and De Holanda, 2009)
Human activity modelling	Low (Chen and Khalil, 2011)

1.5 Objectives of the dissertation

In view of the effective results achieved by Geosimulations for some sectors (e.g. transportation, logistics, telecomms) (Müller and Fischer, 2014), it is time to think **how to improve current Geosimulations and extend them to cover research areas that have been left out of these developments.** The primary objective of this study is to design a generalist Urban Systems simulation methodology and evaluate how it can be applied to different urban planning problems. On this premise, the hypothesis of the thesis is constructed to prove that:

> Time-asynchronous, distributed and intelligent environment based Geosimulations could allow a better description of Urban Systems.

This objective is intended to be addressed from a social and physical dimension to simulate all the interactions and feelings that occur while interacting with the urban environment and help in decision-making processes.

1.5.1 **Structure of the thesis**

The current dissertation proposes a simulation framework called *GeoWorldSim* (Pijoan, 2017) for carrying out the experimentation. The motivation and architectural design of the system is discussed as follows: Firstly chapter 2 reviews the existing trends, strengths and weaknesses in Geosimulation platforms to identify how to model Urban Systems. Secondly, in deep description of the GeoWorldSim core is written in Chapter 3 together with the physical and logical architecture of all the platform tools described in Chapter 4. Finally three applications of GeoWorldSim are detailed in Chapters 5 to 7 applying the platform in three urban problems:

- how to approach IoT communications and their impact (Chapter 5),
- sustainable transport fostering (Chapter 6) and
- power networks resiliency in the city (Chapter 7).

Each of these chapters contains a full description of the problem, solution proposed together with results and specific use-case conclusions gathered. Finally Chapter 8 brings to light conclusions, new challenges and future research areas regarding Geosimulations.

The work here contained, yields a useful tool for urban planners and decision-makers, who can study and develop efficient simulations that can capture the nature of the process involved in their own complex decision-making objectives.

CHAPTER

2

Roadmap to Geosimulations

Societies and territories are interrelated and in constant evolution; even the smallest actions have an impact on the dynamics of both. Change is only noticeable when different patterns become discernible, but before change at the macro-level can be seen, it is taking place at many micro-levels simultaneously. All of these micro-level subsystems interact separately forming a complex network of interactions. In essence, urban dynamics are more than a simple sum of their parts and to understand them, it is necessary to deeply study the individual entities that make up said system.

A clear example is that, although geographical properties are objective measures, as soon as adding the society (human beings) together with their preferences and skills, a territory takes on different interpretations. The slope of a road is an objective datum, but whether this is accessible or not completely depends on the person who has to cross it. That is why urban developments analysis and models need to go into more detail level and consider those components that influence urban dynamics

such as social, demographic or economic factors. If these factors are already complex to understand, these are even more difficult to recreate within a computer system.

This complexity emerges mainly due to its stochastic nature where a large number of variables and personal preferences come into play. Facilities building and urban developments are highly expensive and if the solution does not fit the necessities of the citizens, the infrastructure might need to be upgraded, increasing the cost or even worse, becoming a worthless investment. Therefore, a cheap and flexible method for testing future infrastructures is to use computer simulations supported by advanced mathematical models.

Geosimulations represent a new spatial model approach that allows to develop simulations of urban development bringing together a great diversity of theories and techniques (Sengupta and Sieber, 2007). These represent one of the most innovative attempts to go beyond traditional spatial analyses by understanding urban phenomena as a consequence of the collective micro-dynamics of complex entities that interact with each other. Entities inside a Geosimulation are often represented at the scale of individual homes, humans or everyday objects and in time measures that approach real time. Enriching GIS with MAS (Figure 2.1) merges both the geographical and the socio-economic reality of a territory in a Win-Win relationship making it possible to model a dynamical complex heterogeneous real-world system with its geographic and social components.

Research in Geosimulations has been active in urban applications in the last twenty years (Batty, 1999; Behnisch and Meinel, 2018) where this technique has expanded the range of analyses that can be achieved through simulation. Early efforts in urban development simulations started by using Cellular Automata (CA) models (White, 1998). Initial models had several weaknesses in handling detail, due to a lack of data available at fine-scale spatial resolutions and only being able to create homogeneous behaviour, which made it difficult to model different entity preferences, travel to work costs, and so forth. Following improvements started joining CA with MAS (Torrens, 2003). In these, CA were best used to represent the dispersal of activity and characteristics between discrete spatial units of urban infrastructure and MAS simulated urban population as collectives of individuals with associated behaviours,

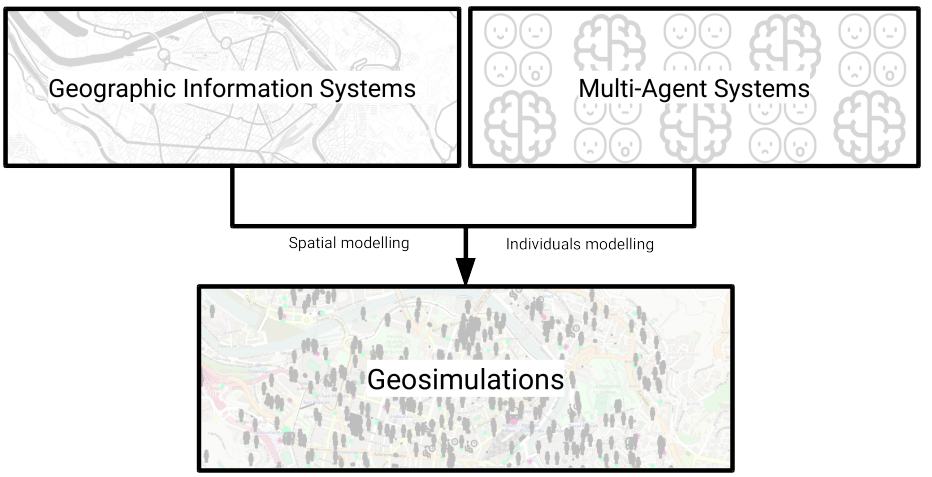


Figure 2.1: GIS with MAS combination to build Geosimulations. (Source: Own research)

traits, mobility and communication. The advance of introducing MAS supposed a great improvement to further explore the evolution of urban developments with heterogeneous behaviour and population.

Since then, the concept of Geosimulations has begun to gain popularity in the literature, facilitated even more by the improvements in computation power and data acquisition. It is noted again that most researchers have created their own software tailored for their needs rather than using or developing a general framework. These explains the huge amount of MAS platforms and Environment Decision Support Systems (EDSSs) found by Rendón Sallard and Sànchez-Marrè (2006). Most of these are not publicly available for inspection of the code. Accessing these platforms openly would have been of incredible help for knowing different methodologies to address Geosimulations.

The modelling urban processes is inherently different to traditional systems. As every LSS, dealing with software that simulates huge amounts of interacting sub-processes causes problems in both the engineering and deploy phases. Mainly have to deal with:

Engineering: Being able to program all the functionality a Geosimulation must provide.

Data harmonisation: Transforming data and objects between different formats and protocols.

Scale: Make use of the computation capacities to run as fast and big as possible.

Real-time: Performing the calculations in acceptable time.

Messaging: Transferring information and commands between separated objects.

Uncertainty: Managing the stochastic nature of Geosimulations.

Knowledge: Digesting the big data of simulation attributes and results into useful information.

Further, Geosimulations need to address several key challenges for urban modelling as identified by Crooks et al. (2008a):

Purpose of the model: Correctly capture the purpose of the simulation.

Theory: The simulation should be based on proven theories and techniques.

Agent and Dynamics: Defining the type, behaviour and interactions between agents.

Calibration and Validation: Fine-tuning the model until it reaches optimal results.

On this chapter the existing Geosimulation works will be reviewed to identify tools and functionality that set the standards for providing Geosimulations. At this point first on the user functionality is analysed, that is, what usability the software provides and secondly on how it is implemented behind.

Reviewing state-of-the-art MAS, a wide ecosystem can be found that models the urban problem using an agent-oriented approach (Rendón Sallard and Sànchez-Marrè, 2006; Kravari and Bassiliades, 2015; Nikolai and Madey, 2009; Change, 2000). In this sense, the extensive review from Nikolai and Madey (2009) identifies more than fifty toolkits for building MAS. Within these, some have been already proposed by Torrens (2010) for being used in Geosimulations development. Most MAS, are strongly linked to their specific use case, would require heavy changes to be used in other contexts and seem to be in a permanent experimental status within an academic domain. However, there exist a few operational and active general purpose

multi-agent platforms that assist with an *all-in-one* toolkit for building Geosimulations and tackling any research area (Marceau and Benenson, 2011). These general purpose MAS, primarily focus on the orchestration of agents providing a run-time environment for executing simulations and allowing experts to focus on modelling rather than programming issues. Above this, said platforms differ on the levels of completion they offer according to the additional built-in characteristics for graphical user interfaces, distributed executions or results processing they integrate.

The most mentioned state of the art general purpose platforms are:

Netlogo (Tisue and Wilensky, 2004): Toolkit designed for simulating complex natural and social phenomena. It enables the user to model any number of agents in a variable-size environment using a simple programming language derived from Logo.

Jade (Bellifemine et al., 1999): A software framework to develop agent-based applications in compliance with the Foundation for Intelligent Physical Agents (FIPA) specifications (Fipa, 2002) for inter-operable intelligent multi-agent systems. It is a middleware to simplify the development while ensuring standard compliance through a comprehensive set of system services and agents.

Repast (North et al., 2005): A widely used agent-based modelling and simulation toolkit with multiple implementations in several languages and built-in adaptive features, such as genetic algorithms and regression.

Gama (Taillandier et al., 2010): A more recently created platform, Gama is a modelling and simulation development environment for building spatially explicit agent-based simulations.

Mason (Luke et al., 2004): A fast discrete-event multi-agent simulation library designed to be the foundation for large custom-purpose simulations and to provide more than enough functionality for many lightweight simulation needs. MASON contains both a model library and an optional suite of visualisation tools in 2D and 3D.

Anylogic (Grigoryev, 2012): A multi-method simulation modelling tool for agent-based, discrete event and system dynamics simulation methodologies. It has

several extension libraries for simulating specific use cases such as health-care, transportation or social dynamics.

UrbanSim (Waddell, 2002): A simulation platform for supporting planning and analysis of urban development, incorporating the interactions between land use, transportation, the economy, and the environment.

REGISTA (Blecic et al., 2009): Reality Emulating Geographical Information System for Territorial Analysis. It joins GIS and CA in a general-purpose platform to model and simulate real-world territorial systems.

MAGI (Blecic et al., 2008): A Geosimulation Infrastructure relevant for strongly geospatially oriented agent-based simulations. MAGI is an effective modelling environment for building and executing simulation models and a class library based on open source components.

OBEUS (Benenson et al., 2001): A Geographic Automata System for urban simulation, a *research-oriented* framework to allow users create GIS simulations through CA. It allows to formulate and code relationships between GIS elements and behaviours for the automatas.

Each of these platforms has its *simulation niche* throughout the literature:

Netlogo and UrbanSim: Found for many urban growth research that models spatial growth patterns and land-use change. The work from Ma et al. (2012),Martínez and Morales (2012) and Albrecht (2005) use them to build Geosimulations integrating urban plans together with economic and social development plans on in order to study the dynamics of the population.

Jade: Conversely, seems to be more oriented to communicating agent system to model urban infrastructure allocation such as security resources, as made by Furtado and Vasconcelos (2006), and flood devices, as Rojas et al. (2017) did.

Gama: Seems to be the *de facto* standard for crowd displacement simulations as most works are focused in people movement, evacuation and urban emergency resiliency simulation. This is the case of the works by Claridades et al. (2016),Carcellar (2017) and Ruiz-Chavez and Salvador-Meneses (2017) where

they use Gama to geosimulate crowd movement inside buildings and urban spaces.

Anylogic: Found mainly for many energy system modelling, optimisation and planning processes in the works by Bahu et al. (2014); Menassa et al. (2013).

Mason and Repast: In the bibliography there are no works in which Geosimulations have been developed using these two platforms. They are mentioned in their technical presentation articles and reviews of the state of the art in Geosimulation techniques, but later the works choose to use another platform or develop their own.

REGISTA, MAGI and OBEUS: Although sounding as promising approaches, these are not actively maintained or their documentation is not publicly available to search how they have addressed Geosimulations.

It is appreciated that many of the these Geosimulations make use of external specialised toolkits. For example, if wanted to simulate the impact of a malfunction on the water distribution network over the citizens, a simulator of the entire water distribution network should be needed. Therefore, if Geosimulations want to cover as many urban scenarios as possible, everything points to the need to integrate highly specialised models. There are already such complex simulators in the literature:

- for fresh and waste water distribution networks, it would be interesting to integrate EPANET (Rossman et al., 2000) and EPASWMM (Gironás et al., 2010; Kamara-Esteban et al., 2017b) software;
- for electric components, it would be desirable to integrate Matlab (Guide, 1998; Kamara-Esteban et al., 2016);
- for transportation, it would be desirable to integrate SUMO (Krajzewicz et al., 2012); and
- for power grids modelling, it would be desirable to integrate EnergyPLAN (Lund, 2011; Østergaard, 2015)

Reading all the above state-of-the-art, it is possible to glimpse the main areas in which Geosimulations are most developed and how the current platforms help in achieving these.

2.1 Operation and limitations

In order to minimise the effort of re-discovering Geosimulations and re-inventing the wheel, the main objective is always to reuse or extend already existing platforms as much as possible. The aforementioned platforms make up the state-of-the-art panorama in terms of Geosimulations. A good analysis of these, allows to discover what functionality they provide to users and which of the platforms does it best. This review will try to extract for reuse all key functionality that Geosimulations should provide.

It must be noticed that the initial revision of the platforms was made at the start of the thesis in 2013. Since then, some of the platforms have been greatly upgraded with new functionality and error fixes. The functionality list contained in this document has been updated to include all the new characteristics added to the platforms but keep in mind that many of them did not exist back in 2013.

Complex simulation platforms tend to have steep learning curves and developers must be familiar with the underlying abstractions or make an effort in understanding those concepts. This entails that, depending on how large the framework is, getting to know it in depth and the development of the necessary extensions time can exceed the time required to develop a new custom one. It is here where the access to the source code of the platform, technical reports and descriptive documentation, play a vital role.

2.1.1 NetLogo and Anylogic

As soon as starting to dive in the source code and documentation it is quickly discovered that two of the above identified platforms are not open-source (**NetLogo** and **Anylogic**). This does not imply that they can not be tested (their characteristics will continue to be analysed) but obviously discards them for reuse. It was a hard blow having to discard Anylogic since it is probably the most mature and complete platform in the market for Geosimulations. Anylogic surpasses all the expected functionality of Geosimulations and seems to really be the leader in its field. It is also true that this maturity has been achieved by orienting Anylogic into a cloud proprietary

product that fulfils the main needs of the industry. Although it contains a free trial period that has been used to test some of its cloud simulations, there is no public information about the purchase costs of the platform.

2.1.2 **Jade**

Following the above order, Jade was next. Looking more in detail at its functionalities, it is important to distinguish two opposite approaches found:

Agent-Based Simulators: All platforms except Jade are so called *Agent-Based Simulators* conceived for creating, analysing and experimenting with artificial worlds populated by agents that interact in non-trivial ways. These deploy a simulation infrastructure that schedules the execution steps of each agent (called ticks), generally following a behaviour in which do not need to wait for the arrival of messages or synchronising with others.

Agent System Builders: By cons, Jade is an *Agent System* framework intended for building peer-to-peer agent systems but with no simulation infrastructure. In other words, Jade is a powerful middleware ready model communication between distributed agents across machines but does not have an agent scheduler, nor a notion of a *clock*. Jade agents are generally scheduled and synchronised by message sharing itself instead of deploying an execution layer. This is more typical for cooperation and conversation processes in which agents need to constantly interchange complex messages and the execution steps are event-driven, that is, subject to the arrival and sending of messages. Within Jade platform, creating high level simulation controls are left for the programmer to code.

This makes Jade the most complete platform in terms of communication functionalities but also the one that more additional development would need for extending it to model Geosimulations.

2.1.3 **Repast**

Platform that started as a small scale simulations platform based on Java programming language. Repast is a complete framework for running general purpose agent-based simulations which includes classes for geographical and network functions. In 2010 the Repast community launched RepastHPC (Collier and North, 2013) an agent-based modelling system intended for large-scale distributed computing platforms Lopez-Novoa et al. (2015). This new version implemented from scratch the core Repast concepts but using C++ language in search of greater performance and to work in parallel distributed computers. RepastHPC offers a suite facilitating creation, execution and evaluation of agent-based models in distributed computers. However this approach is for creating simulations in which agents do not need to share any messages since RepastHPC does not provide communication support for remote agents.

2.1.4 **Gama**

Gama, as stated by their developers Taillandier et al. (2010), emerged to fill the gaps other platforms had for building spatially explicit agent-based simulations. For this reason, a large part of its efforts have focused on providing advanced spatial operations to both agents and the environment. It is undoubtedly one of the most complete and advanced platforms for the simulation of generic spatial models. Gama is constantly being enriched with new functionaltities but it seems that future development is going to left aside taking the platform to the cloud paradigm or being able to develop distributed simulations (Grignard et al., 2013).

2.1.5 **Mason**

This is a robust and fast simulation tooltkit programmed in Java. Mason is a fast, easily extendable, discrete event multi-agent simulation library designed to serve as the basis for a wide range of multi-agent simulation tasks. Being a library, and not a complete framework, makes the transition from ideas to working simulator

more difficult than others. GeoMASON (Sullivan et al., 2010) is Masons's version dedicated to achieving Geosimulations. Comparisons carried out by Barbosa and Leitão (2011); Railsback et al. (2006) agree in entitling GeoMASON as the fastest toolkit for building MAS. However its inner programming environment is complex and its spatial operations support are still being improved. GeoMASON still has some pending limitations regarding GIS and distributed simulations as identified by Luke et al. (2018).

2.1.6 **UrbanSim**

Finally, UrbanSim is a urban-growth linked platform that combines huge data-sets with built-in models for: household and employment relocation; location choice; real state pricing and renting; real state development; and travel. This seems great base to be extended to accommodate other Geosimulation types. However it does not seem to be an multi agent system. When using UrbanSim, users parameterise the built-in models rather than defining agents, their behaviours and goals. Looking at the code there is no agent class and the selected optimisation model would be the active processes that makes the calculations, instead of having independent agent elements competing or cooperating. In addition, UrbanSims further development is conditioned by its programming language: the inability of Python to use more than one computer processor at a time. Python contains a Global Interpreter Lock (GIL) (Beazley, 2010) which protects access to objects, preventing multiple threads from executing Python byte-codes at once. This lock, necessary mainly because Python's memory management is not thread-safe, will not be corrected in the short term (Python user community, 2018). In the era in which computers have more and more facilities to execute code in parallel the extension of UrbanSim to other Geosimulations can lead to reaching a bottleneck or not being able to achieve the expected performance. In this sense, Awaludin and Chen (2007) already researched about adding parallelism to UrbanSim introducing several approaches and all the problems they found. Although achieving very promising results, they highlight the infeasibility of going large scale due to the high memory it would require. Awaludin

and Chen (2007) propose several recommendations to be carried out for improving UrbanSim parallelisation but it seems that these have not been still added and no plans of doing is mentioned.

2.1.7 **Identified functionality**

Parallel to inspecting the above platforms' code, each was launched and operated to try to grasp what options these offered to the users, reaching a minimum common denominator of capacities that should be standard in all Geosimulation. MAS' complexity is intimidating, with dozens of classes and hundreds of methods, and makes it difficult to precisely determine all the possibilities offered by each tool. Even so, Table 2.1 displays the user functionality offered by the selected general purpose MAS. Mainly, the capacities analysed are:

- Distributed simulations.
- Built-in graphical interface for viewing the simulation course.
- Visual configuration of a simulation without programming knowledge.
- Results are processed into charts and graphs.
- Web based platform.
- Multiple user management.

Two development philosophies can be appreciated: platforms that try to integrate all the *expected to be helpful* functionality and those leaving some important parts out for programmers to implement as external modules or plugins (e.g. Jade simulation Graphical User Interface (GUI) or Repast and Anylogic distributed communications).

Trying to put into the simulator as much functionality as possible is not always the best option. While it is true that adding extra capabilities to a platform is a good way to attract users and create more complex scenarios, additional load makes simulations lose speed and performance. Having to render the status of an scenario during execution, requires having a process that iterates all the agents at the end of each simulation step and draws their situation. Moreover, if charts and graphs need to be rendered, the process does not only need to iterate all agents but to run some

Table 2.1: User functionality of general purpose MAS. (Source: Own research)

User functionality	**NetLogo**	**Jade**	**Repast**	**Gama**	**Mason**	**Anylogic**	**UrbanSim**
Open Source	✗	✓	✓	✓	✓	✗	✓
Programming language	NetLogo	Java	Java, C++	Java, Gaml	Java	Java, UML-RT	Python
Distributed simulations	✗	✓	✓	✗	✓	2	✗
Simulation GUI	✓	1	✓	✓	✓	✓	✓
Visual programming	✓	✗	✗	✓	✗	✓	✓
Charts	✓	1	✓	✓	✓	✓	✓
Web based	✓	✗	✗	✗	3	✓	✓
Multiple users	✗	✗	✗	✗	✗	✓	✓

1. GUI for visualising the execution of a simulation is not a native built-in feature. However Jade implements mechanisms for reading the data and messages transmitted between agents for users to program visualisation tools delegating in external Java GUI packages.
2. Repast and Anylogic platforms can run distributed simulations but relying on external messaging tools (e.g. Active MQ) that are not directly built-in the simulation core. Using message queues these two platforms can run distributed agents that synchronise through waiting for others' messages.
3. Mason, despite not being a core functionality, offers the possibility for taking the agents to the web environment by users programming their own Java Servlets that communicate with the platform.

calculations with these. Depending on the information to be charted, computation can take more time than the one agents needed to execute the step of the simulation. This is the reason why it is necessary to find a balance between the core functionality that must be integrated within the simulation engine and other secondary processes, which are not compulsory for the simulation to wait for their operations, that can be delegated in external modules.

2.2 Functionality implementations

The implementation of each platform has been analysed also to understand how these implement the above functionality. As mentioned, five of the platforms (Jade, Gama, Mason, Repast and UrbanSim) are open source and thus, their inner code can be inspected in search of how they have achieved Geosimulations. The remaining two (NetLogo, Anylogic), despite being non open source are present in much of the literature on Geosimulations so it seemed essential to take them into account.

2.2.1 Intermediate languages

One of the first features interesting to mention, is how some platforms tend to abstract users from the real programming of the agents. Using high-level languages, they facilitate usage to non programmers and novice users through structures and primitives that greatly reduce programming effort. A high-level language emerged to standardise the behaviour modelling of agents, was the *3APL* (Hindriks et al., 1999), an experimental language created to support a way of programming goal oriented agents. However agent platforms seem not to have implemented it and some have opted for using other high-level languages or developing their own. This can be found in NetLogo, using the Logo (Tisue and Wilensky, 2004) general purpose programming language; Gama, with their developed Gaml (Taillandier et al., 2010); and Anylogic, with the real time UML standard UML-RT (Fischer et al., 2001).

2.2.2 Communication

Communication between agents has been an active research area. The MAS community, has made great efforts in designing open systems and universal communication mechanisms for them to speak together. In this sense, there has recently been increasing interest in incorporating organisational concepts into MAS and designing organisation-centred designs (Rodrıguez, 2001). When referring to agent communication, it is essential to talk about the FIPA. This is an international organisation dedicated to promoting the industry of intelligent agents by developing specifications for interoperability among agents and applications. The FIPA Agent Communication Language (ACL) messages are widely used when communicating distributed agents and processes (Luck et al., 2012) and FIPA-compliance is a reference point for many researchers since it guarantees the agent cooperation and interoperability through well-tested protocols. In this sense, Jade is a full FIPA-compliant platform and only NetLogo and Gama have plugins for integrating FIPA messages.

2.2.3 **Distribution**

Communication is the basis to create distributed simulations which run on more than one computer at the same time and in synchronisation. Most of the popular agent platforms, started to be developed twenty years ago when computational power was still scarce. In order to achieve computationally intensive simulations with many agents or long run times, some decided to improve the execution in a single Central Processing Unit (CPU) architecture and others concentrated large part of their efforts on getting simulations running distributed across several computers. Distributed MAS, allow agents to be moved and controlled among machines synchronising all to work together. Distributed simulations and how to synchronise them have been a key factor to improve system scalability.

Abar et al. (2017); Kravari and Bassiliades (2015) performed two studies respectively reviewing which platforms support distributed simulations and their performance achieved. In this line, Jade, Mason and Anylogic have proven success in implementing large-scale distributed simulations by using two approaches. On the one hand, Jade and Mason determine a *master* simulation or control which deploys additional functionality in order to federate the remaining *slave* simulations. On the other, Anylogic uses the publisher-subscriber mechanism to provide two ways of coordination. If the distributed simulations are not time regulating nor time constrained, then these should not add any special time coordination services. Conversely, if time regulated and/or time constrained federate regulation is needed, Anylogic uses its *next event request* (Borshchev et al., 2002) service calls that synchronises all the distributed systems. Nevertheless, although being a very useful feature for simulating real world distributed mechanisms, improvements in hardware have made distributed simulations no longer necessary for simulating large number of agents. Current computing power and cloud computing capabilities have enabled single CPU platforms to scale to computationally intensive simulations without needing to go distributed.

2.2.4 Coordination

Coordination of the execution in a MAS is designed to be either synchronous or asynchronous. In synchronous models, all agents are assumed to change simultaneously. The calling order of the objects has no influence in this mode but conflicts can arise when agents compete over limited resources. Conversely, with asynchronous timing, agents change in turn, each observing the reality left by the previous agent and mentioned conflicts are therefore resolved. This order of updating agents (often, but not necessarily, random) is critical as it makes the model results stochastic. Traditionally in MAS, each simulation step is known as a *tick* and within each, usually all agents are given time to perform their behaviour. A simulation is a continuous loop in which periodically calling the `tick()` function, advances the agents' situation. Furthermore, agents traditionally have a `tick()` function to be called within the *tick* loop and serving as the starting point for each decision process in their behaviour. On the one hand, in NetLogo or Gama time passes in discrete steps with the same length and all agents are *ticked* in said steps. On the other, in Jade, Mason or Repast agents are event-driven, that is, programmed to be *ticked* at some time in the future.

2.2.5 Geographic operations

Finally, geographic operations are also an important functionality to include in a Geosimulation engine. In this sense, although most platforms provide some simple physical layer to situate and give geometries to agents, only Gama and UrbanSim add rich support for complex geographical operations at agent level (e.g. moving an agent inside a geometry, computing a shortest path between two points of this geometry or on a network). In this sense, NetLogo and Mason can extend their GIS computation through third party plugins providing the ability to load vector GIS data (points, lines, and polygons) and raster GIS data (grids) into the model (Russell and Wilensky, 2008). On a different approach other, UrbanSim contains a novel Temporal-GIS model that adds not only geographic features but also a location in time to agents (Shaw and Xin, 2003). These time proportions are used in a temporal GIS database to time-stamp every datum to give it temporality. The valid time is the time period for which a

spatial phenomenon is considered to be of the foremost concern to the GIS users, because it is the time-based information that supports spatio-temporal analysis.

Identified characteristics set the starting point for designing the Geosimulations core. Table 2.2 summarises all these points.

Table 2.2: Simulation core characteristics. (Source: Own research)

Core functionality	**NetLogo**	**Jade**	**Gama**	**Mason**	**Repast**	**Anylogic**	**UrbanSim**
Simulation infrastructure	✓	✗	✓	✓	✓	✓	✓
FIPA messaging	1	✓	1	✗	✗	✗	✗
Large-scale	✗	✗	✓	✓	✓	✓	✓
Distributed executions	✗	✓	✗	✓	✗	✓	✗
Asynchronous coordination	✗	✓	✗	✓	✓	✓	✗
GIS Integrated	1	✗	✓	1	✓	✓	✓
Checkpoints	2	✗	2	✓	✗	✗	✗

1. Functionality that is not native of the simulation core but can be added through plugins.
2. Platforms that although not having an explicit checkpoint recovery method, do not seem to save much information from previous states of simulation steps. Exporting the entire simulation at a certain step (e.g. creating a Shapefile or eXtended Mark-up Language (XML)) apparently could work as a way of re-starting a new simulation at that checkpoint (loading the evolved Shapefile as if it was the initial state).

2.3 In search of the reusable

In view of all these different approaches and limitations, it becomes clear why the initial review of MAS discovered so many platforms developed from scratch to serve their specific scientific purposes. It would have been really successful to reuse one of the platforms and extend it to cover new Geosimulations. However when the platforms where firstly analysed, at the beginning of this work in 2013, found drawbacks and time needed to extend the platforms required a bigger effort than developing a new general purpose toolkit.

The most important decisions that conditioned discarding the existing tools were:

- Geosimulations should take advantage of all the computational power offered by multi-core computers, distributed simulations and the cloud computing paradigm.

- The platform should be user-friendly and try to provide visual non-programming mechanisms for building the Geosimulations.
- It should be possible to provide complex Geosimulations through the web to bring them closer to any audience and avoid user device limitations (as the computation would be performed in the server).
- Geosimulations should integrate distributed systems (e.g. sensors, IoT, etc.) into their environment in an efficient and transparent way.

The review from Nikolai and Madey (2009) highlights that the main programming language most models have adopted is Java. Moreover, the best platforms identified (Anylogic and Gama) are built upon Java. However, it is believed Java hauls some limitations that could be a ceiling for the contribution of this work. This thesis seeks to achieve effective Geosimulations that can be distributed in separate machines and must integrate external expert models (e.g. water models, power models, transportation models, etc.). The platform must also offer the whole life-cycle of creation, simulation and result knowledge extraction across the web. These features are not something imposed by the objectives of this work, they are also mentioned in some of the above platforms' future work agenda (Luke et al., 2018; Awaludin and Chen, 2007; Grignard et al., 2013) but do not seem to be feasible in the short term.

Thus, this thesis decides to make a commitment in building a platform from scratch using C++ programming language. As Lextrait (2018) states, ICT giants tend to program their products and utilities using C++. Java provides higher-level abstraction than C++ and uses a Java Virtual Machine (JVM) to be executed in any operating system (Meyer and Downing, 1997). The JVM does some automatic optimisation of the written code but, in contrast, requires a higher memory usage and does not allow so fine-grained memory management. C++, although requiring specific compilation, allows a higher memory efficiency and performance due to its lower-level programming which is much closer to the operative system. This is also supported by the direction adopted by two well established simulation platforms, Repast (with RepastHPC) and Jade (with Jade's SIM++), who decided to port their models to C++ in order to achieve better performance and the ability to run distributed.

Summarising, Table 2.3 describes how desirable it would have been to reuse the tested platforms, their needed extensions and the why have been discarded.

All the work done in reviewing the above platforms does not fall on deaf ears. Knowing the interiors of the platforms allows to think of a mental planning for designing a new platform that combines identified success stories together with the solution to the deficiencies found. As will be further explained, the conclusions other platform developers have made, provide an important starting point for this research work.

Table 2.3: General purpose MAS' strengths, limitations and reasons to be discarded. (Source: Own research)

	Strengths	Limitations	Discarded reason
Netlogo	Easy use and friendly programming language (NetLogo)	Small-scale and educational purposes oriented	Not suitable for large-scale simulations.
Jade	It is a complete FIPA messaging framework	Has no simulation infrastructure	Would require creating the entire MAS except for the communication infrastructure.
Gama	Best open source platform. Proven success in GIS and LSS	Developed in Java	When this thesis work started it was not as mature as today. Now it would probably be the way to follow and fit it as the ecosystem simulation core.
Mason	General purpose MAS with simulation infrastructure.	Not web oriented and limited GIS operations.	Adding GIS functionality and taking it to the web would require big effort.
RepastHPC	MAS for high performance distributed simulations coded in C++	Not web oriented and limited GIS operations.	Adding GIS functionality and taking it to the web would require big effort.
Anylogic	Best cloud MAS with big amount of ready-to-use models for main industry needs.	Not open source	There is no access to the code to implement the required extensions.
Urbansim	Cloud based MAS with visual parameterisation of the simulations.	Developed using Python	Would require re-implementing the entire execution core to asynchronous and multi-threaded.

CHAPTER

3

Geosimulations design and implementation

Based on the collected evidence, **it was decided to design a new platform, GeoWorldSim (Pijoan, 2017), for giving support to all the life-cycle of Geosimulations.** This is the result of all research and development made throughout this dissertation and one of the biggest contributions. GeoWorldSim captures all the state-of-the-art functionality a Geosimulation must provide and improves some limitations found. Shaping urban systems through Geosimulations allows a bottom-up modelling to streamline and complete many of the geographical analysis that are done regularly. It makes possible dividing a problem into sub-problems and assigning an agent to each of those sub-problems. All this process requires reversing the programming flow of traditional analysis, building the models with a decentralised approach. The fact of modelling firstly the behaviour of each individual entity for after analysing and building logical aggregations, achieves significant advantages over traditional models:

Flexibility: Geosimulations speed up many of the usual spatial analysis. Given its agent-based nature, once the logic of an agent is modelled, it is easy to automatically add more instances (more agents) that replicate the process. This provides a natural framework to fine-tune the complexity of the agents: behaviour, degree of rationality, ability to learn and evolve, the rules of interactions, etc.

Distributed control: Unlike traditional spatial analysis, the Geosimulations are distributed and concurrent processes. Each agent runs in its own thread parallel to the others, not being able to predict the order in which they are going to act. In the real world there is no central control in charge of deciding the order in which each individual must carry out its activities, but each acts independently. It is the the decisions as a whole of all the agents that have an effect on the environment. In short, the Geosimulations provide the following features in terms of execution:

Concurrence: It is necessary to take into account that agents are going to run from parallel, so some critical sections and resources need concurrency management.

Performance: Separating processes into agents involves that, in case of increasing the number of tasks, the time needed for the consecution of the process does not increase linearly.

Indeterminism: The order of execution of the agents is totally random so depending on the type of simulation to be carried out, the final result may have slight variations. This characteristic can suppose a negative property for some researches if they want simulations to be reproductible obtaining always the same result.

Spatial thinking: As already mentioned, agents must be spatially explicit. This implies two concepts, on the one hand they must become entities located in a georreferenced two-dimensional or three-dimensional environment. Although years ago there were agents that simply internally contained a variable x and y or the environment was divided into cells, it is now necessary for the environment to locate the agent, which should be georreferenced in every moment

so that space operations can be carried out with their location. In addition, on the other hand agents must also have spatial capabilities in their decision processes.

Discrete objects: Analysis built with a top-down approach generally make use of spatial partitions either in the form of cells or by groupings of elements for which it is assumed that all the entities contained within are homogeneous. The fact of programming the simulations modelling the entities from bottom to top through agents with their particularities, it achieves results with a behaviour that is more natural and closer to reality.

Major Programming: This paradigm allows individual programming and configuration of each of the different individuals that will interact with the environment, whether people, vehicles, buildings, street lighting, etc.

More accurate: The exhaustive definition of the main characteristics of each agent, as well as the modelling of behaviour and interaction between them allows a more dynamic, natural and close to reality model.

Dynamic environment and quasi real time: Another of the great advantages and possibilities that Geosimulations provide is the use of dynamic environments and measures that approach the real time. Although, traditionally, static environments have been used which contained a frozen snapshot of a territory at a given moment. Geosimulation environments are also born from a snapshot but evolve over time. Depending on how the agent environment is developed, this can be directly connected to the real world, so that the changes in this last one are reflected in the simulation.

Reading how other works have structured their simulations, it was decided to build GeoWorldSim simulator on top of four basic pillars depicted in Figure 3.1:

Agents: Explained in Section 3.2, these are all elements that interfere in a simulation.

Environments: Explained in Section 3.6, first level entity for modelling collective behaviours resultant of individual agents' aggregations.

Skills: Explained in Section 3.3, abilities of agents to perceive their world and propagate actions in it.

Behaviours: Explained in Section 3.4, decision model for agents to orchestrate their skills and achieve their goals.

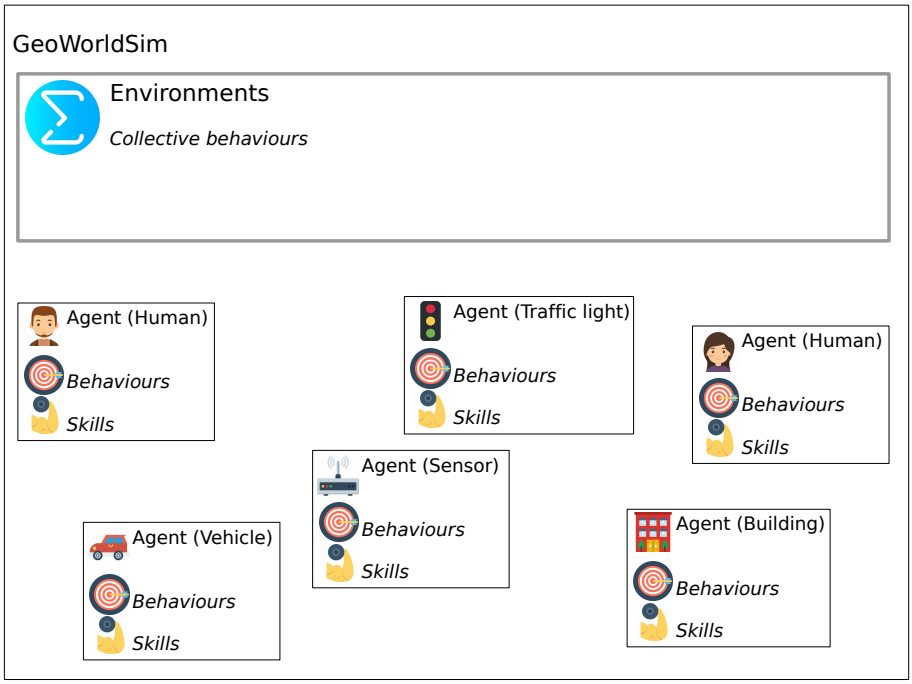


Figure 3.1: GeoWorldSim simulator's main elements: Agents, Environments, Skills and Behaviours. (Source: own research)

Additionally, during the research, a strong effort was made in compartmentalising each sub-necessity of the platform in modules. The simulation core should only focus on carrying out the evolution of a scenario, working with standardised formats and not being in charge of custom visualisation of the results. Further functionality should be taken out of the simulation engine's scope and delegated to other pieces of software.

This is put into practice by splitting the GeoWorldSim platform in ten modules (simulation core and nine secondary modules) that build the architecture. The

ecosystem detailed hereby allows to address the needs of Geosimulation platforms by joining traditional concepts and mechanisms with some novel perspectives and techniques. GeoWorldSim through all its components currently provides all the requirements contained in Table 3.1.

	GeoWorldSim
Open Source	✓
Programming language	JSON-LD, C++
FIPA messaging	✗
Large-scale	✓
Environment centric	✓
Standard ontology based	✓
Asynchronous coordination	✓
GIS Integrated	✓
Checkpoints	1
Distributed simulations	✓
Simulation GUI	✓
Visual programming	✓
Charts	✓
Web based	✓
Multiple users	✓

Table 3.1: GeoWorldSim characteristics. (Source: own research)

In this chapter, the implementation of the simulation core will be fully detailed while the remaining modules are left for chapter 4. In following sections, Section 3.1 describes the programming concepts on which the software has been built, Section 3.2 writes about the agents of the simulation, Section 3.3 describes the skills these agents can have, Section 3.4 details how to program an agent's behaviour and Section 3.6 describes in depth the technical and methodological aspects of GeoWorldSim's core environment.

3.1 Software platform

It was decided to be build the platform using Qt framework (Qt Company, 2018a), an object-oriented application development framework based on C++ programming

language. C++ is used in many everyday contexts, with key strengths in software infrastructure and resource-constrained applications, such as desktop applications, computing servers or performance-critical applications. In contrast with object-oriented languages, traditional procedural-oriented languages (like C or FORTRAN) suffer notable drawbacks in creating reusable software components. Procedural programs are made up of a series of step by step instructions, with function as the basic unit. They make use of functions prioritising logic rather than data, first figuring out all the functions and then thinking how to represent data. Conversely in object-oriented programming, importance is given to data rather than just writing variables and instructions to complete a task. An *Object* is an entity or idea that is described, implemented and reused in a program and provides the following features:

Classes and functions: Classes are a definition of objects of the same kind. In other words, a class is a blueprint, template or prototype that defines and describes the static attributes and dynamic behaviours common to all objects of the same kind. Using these templates, an instance is a realisation of a particular item of a class. In other words, an instance is an instantiation of a class. All the instances of a class have the same attributes, as described in the class definition. Classes allow hiding of attributes by declaring them public or private. Private attributes can not be used outside of a class, avoiding outer code to change important attributes. Functions declared inside classes have access to these private attributes for the programmer to enable a controlled way of interacting with private attributes.

Inheritance: Is a powerful feature of object-oriented programming languages which helps in organising classes into a hierarchy and enabling these classes to inherit attributes and behaviour from above in the hierarchy. Inheritance describes an *IS A* relationship where the parent class can contain common attributes/behaviour and later be specialised in child classes. It does not work backward, that is, a parent will not have properties of the derived class. Inheritance is a mechanism for code reuse and can help in reducing duplication of code. However, trying to force inheritance and adding multiple levels of abstraction can lead to writing unnecessary code. It is therefore important to

firstly identify common attributes and behaviour in the entities that are going to be modelled and based on that define a suitable hierarchy.

Polymorphism: Is the concept that there can be many different implementations of an executable unit and the difference happens all behind the scene without the caller awareness. Polymorphism allows computer systems to be extended with new specialised objects being created while allowing current part of the system to interact with a new object without concern for specific properties of the new objects. Using polymorphism a behaviour defined in a parent class can be overridden by a child writing a custom implementation of the method.

Despite this efficient run-time environment, C++ general-purpose nature lacks of additional utilities in certain event-based domains. This is where Qt provides some extra power, by combining the speed of C++ with the flexibility of its own Qt Object Model.

3.1.1 Objects

GWSObjects are the heart of every GeoWorldSim element, that is, their base Class. They are built on top of Qt's QObject model (QObjects) () inheriting really powerful features from Qt. The only base attributes all GWSObject share in common are:

Id: Unique identifier which is used for indexing and identification. These are generated by combining the simulation identifier, the object type and an incremental number e.g. *SIM4-Agent-845*, *SIM4-Road-523* or *SIM4-Sensor-816*.

Class type: Object's class type that can be found dynamically via its corresponding `metaObject()`.

These two attributes are what define and identify any GWSObject in the platform. The alluded native Qt functionalities need to be activated during compilation. For this, GWSObjects must declare a *Q_OBJECT* macro in the object's class. Even if not needed, the use of this macro is strongly recommended in all sub-classes of QObject, since failure to do so may lead certain functions to exhibit strange behaviour. The great advantages of Qt are detailed below.

3.1.2 Parent-child relationships

GWSObjects organise themselves in object trees, that is, when creating a new GWSObject another one can be appointed as a parent. The newly created object will automatically add itself to the parent's `children()` list. This is important in terms of memory management and multi-threading since a parent takes ownership of its child objects by on the one hand storing all them in the parent's thread and, on the other hand, by automatically deleting all children in the parent deletion. By cons, an object with no parent will need to be deleted manually. Qt also uses these parent-child relationships for user interfaces where hiding a parent hides the children or for key and touch events that are propagate to the parent if the child does not handle them.

3.1.3 Meta-properties

Qt provides a sophisticated property system to create and access on-the-fly class attributes without previously having declared them. This is done through key-value property storage which can be accessed through two methods: `setProperty(name, value)` to store a value and `getProperty(name)` to retrieve it. Qt defines a QVariant class (Qt Company, 2018c) which is a union type for the most common data types. Using QVariants, a meta-property holds a single value of a single `type()` at a time such as boolean, integer, double, char, string, date, time, colour, etc. Additional types can also be hosted inside meta-properties with the `Q_DECLARE_METATYPE` primitive.

To retrieve the type of a QVariant and make transformations, all contain a `convert()` method, many `toT()` functions (e.g. `toInt()`, `toString()` or `toDouble()`) and a `canConvert()` to check whether the type can be transformed to another type. If the conversion is possible, these `toT()` methods return a copy of the stored object that is, the original object inside the property remains unchanged. On the other hand, if asked for a type that cannot be generated from the stored type, the returned value will be different depending on the type.

Meta-properties are a very useful way for serialising and deserialising objects. Instead of writing a save and load code for every single class, a generic serialisation mechanism has been designed using QObject's introspection/reflection and the *Factory design pattern*. For every GWSObject, creating meta-properties on-the-fly, there is no need to know all attributes an Agent has nor the `getters()` and `setters()`. More about the GWSObject Factory is described in Section 3.2.1.

3.1.4 Signals and slots

The main new mechanism that enables bringing asynchrony to the GeoSimulation world. Qt objects have this powerful mechanism for seamless object communication called signals and slots which is included in the GWSObjects and can be used in a direct way. The signals and slots are a key feature of the Qt Framework and probably the part that differs most from other frameworks. **This dissertation proposes deferring intra-agent communication through the use of Signals and Slots to release resource locking and create interactions more similar to those of the real world.**

GWSObjects can declare signals, which are function names that can have parameters but have no implementation. A signal can be emitted anywhere and anytime through the code by using the `emit` reserved word followed by the signal name and the parameters to be sent in the signal. Conversely slots are declared as normal functions and must match the arguments that are sent by the signal. Combining both, GWSObjects can connect a signal to a slot (as a publisher-subscriber mechanism (Eugster et al., 2003)) with `connect()` and destroy the connection with `disconnect()`. Instead of agents calling functions, GWSAgents emit signals to spread changes in a way similar to what is used in distributed simulations messaging.

This is a really central mechanism for modelling how humans perceive external stimuli created from other entities who broadcast a message without needing to know the receiver. A signal can be connected to 0 to N slots so many entities can listen the message emitted from one object. Qt internally has an *event loop* (further explained in Section 3.6.7) that asynchronously attends the arrival of signals, queues the slot invocations and executes them. Within a slot, there is a special static method called

`GWSObject::sender()`, which returns a pointer to the object that activated the signal, if automatically called through a `connect()`; otherwise it returns 0. This pointer is valid only during the execution of the slot, from this object's thread context and will become invalid if the `sender` is destroyed, or if the slot is disconnected from the `sender`'s signal. Getting access to the `sender` is crucial when many signals are connected to a single slot in order to know who triggered the code.

For a better understanding, Listing 3.1 describes and example for modelling `Clients` waiting for a reservation counter `Indicator` that announces whose turn it is. The counter `Indicator` announces the next turn with a signal and the `Client` whose turn is called, goes to the counter.

Finally, whenever a object is deleted, it will emit a `destroyed()` signal which can be caught to avoid dangling references to deleted GWSObjects.

```
public Indicator { // Reservation counter attending clients
    Q_OBJECT // Needed macro
    public:
        Indicator(); // Constructor
    signals:
        void nextClSignal( int turn_number );
}

public Client { // Person waiting to be attended
    Q_OBJECT // Needed macro
    public:
        Client( int my_turn ); // Constructor
    slots:
        void nextClNotification( int turn_number ){
            if( turn_number == this->my_turn ){

                Indicator* rs = dynamic_cast<Indicator*>( QObject::sender() );
                rs->disconnect( rs , &Indicator::nextClSignal ,
                this , &Client::nextClNotification );
            } else {
                // Do nothing
            }
        };
}

int main(){
    Indicator* rs = new Indicator();
    Client* w1 = new Client( 111 );
    Client* w2 = new Client( 233 );

    rs->connect( rs , &Indicator::nextClSignal ,
        w1 , &Client::nextClNotification );
    rs->connect( rs , &Indicator::nextClSignal ,
        w1 , &Client::nextClNotification );

    emit rs->nextClSignal( 111 );
    // Client w1 will head to the counter
    // Client w2 will continue waiting
}
```

Code 3.1: Example code of signal and slots declaration and connection. (Source: own research)

3.1.5 Implicit sharing

Not all Qt's functionalities show to be an advantage in all cases. Qt uses an optimisation called implicit sharing for many of its value classes to maximise resource usage and minimise copying. Implicitly shared classes are both safe and efficient when passed as arguments, because only a pointer to the data is passed around, and the data is copied only when a function writes to it, that is, *copy-on-write* technique (Hitz et al., 2005). This implies that all implicitly shared classes in Qt use atomic reference counting to ensure the integrity of the data being shared and in how many objects it is being referenced. E. g., when creating a QString `sNew` from another QString `sOrig`, the created `sNew` is not a copy of the initial element but a shared reference to `sOrig` which will have its reference counter to 2. When a different value wants to be given to `sNew`, then a deep-copy of `sOrig` is performed, its counter is set back to 1 and `sNew` finally becomes a new space in memory with the newly assigned value.

Usually implicit sharing and multi-threading are incompatible concepts, because of the way the reference counting is typically done. Shared objects are often passed between threads so, if a thread emits a signal, passes a QString to a target slot on another thread and quickly changes that QString, there is a possibility that a task switch could occur somewhere in this time frame and cause heap corruption. Although, Qt's atomic reference counting ensures the integrity of the shared data between different threads, it does not guarantee the value to be thread-safe. Proper locking (mutexes) must be used when sharing an instance of an implicitly shared class between threads.

Big discussion can be found about the benefits and drawbacks of different approaches for managing implicit sharing and how these affect the performance(Sutter, 1999). Mostly all experts agree that atomic reference counting is considerably slower than normal integer reference counting which is to be expected; since it constantly needs to read/set the reference count in memory and can only go as fast as the memory bus allows. However, even with the extra cost of the atomic increments and decrements, implicit sharing using atomic reference counting is more memory efficient than not sharing at all. The decision was to combine best of both by using Qt classes in most of the code but falling back to base C++ types and data structures

in cases where data is moved through threads and therefore using lockers would carry performance issues. One of this cases is the GWSUnit class (developed to represent resources and magnitudes) which is exchanged between objects belonging to different threads and whose QStrings have been changed to char*.

There is no way for comparing the performance between the original and the adopted solution since the original implicit sharing crashed any attempt to running the software.

3.2 Agents

In all MAS, agents are the actors that perform in the simulation. These actors are pieces of software that contain a certain behaviour, objective and state. MAS agents are programmed to operate autonomously without the direct intervention of any kind and have control over their actions and internal state. Additionally, according to their behaviour they can have functionalities of interacting with other agents and humans, perceiving their environment and responding in a timely fashion to changes occurring therein and performing a goal-directed behaviour by taking the initiative.

Inside GeoWorldSim, GWSAgent is the class for modelling entities and every real world tangible thing is an agent. This class is derived from the GWSObject parent skeleton that contains the basic mechanisms above described. Additionally, all GWSAgents have a behaviour block and a set of skills even though they may be empty. The internal structure of a GWSAgent is described depicted in fig. 3.2. Agents can be categorised in two different groups whether they have a proactive behaviour or they just react to stimuli.

Autonomous agents: Constitute the traditional agents in a MAS. They are proactive actors in the simulation and are programmed to follow a certain behaviour. Autonomous agents are scheduled to emulate a real world entity as deep as the necessary level of detail of the results. They will contain a block of behaviour that will be executed on each call of the `tick()` function. To access and propagate their actions, autonomous agents will make use of their skills.

Passive entities: On the other hand, Passive Entities are those agents who, although having some *logic* inside, just react to the actions of others. Namely, they provide or receive information from agents and other passive entities. Note that passive entities can make use of skills to propagate their reactive behaviour but this behaviour will be invoked by outer active agents and not through the `tick()` function.

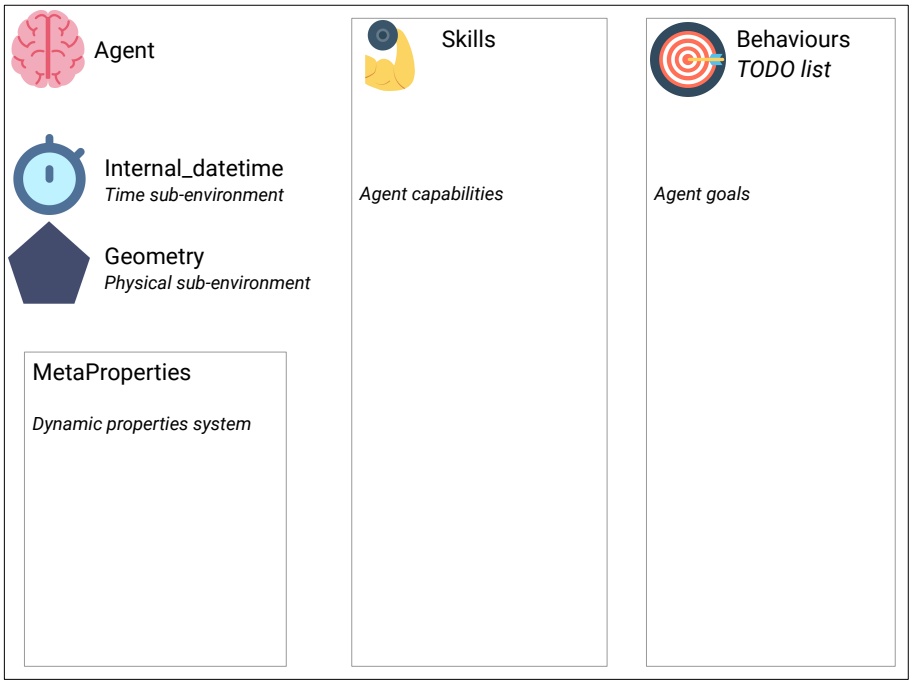


Figure 3.2: Elements that compose a GWSAgent. (Source: own research)

3.2.1 Agent Ontology and Factory

Ontologies can be defined as explicit formal specifications of terms and relations of a certain domain (Gruber, 1993). An ontology defines a common vocabulary

for researchers who need to share information in a domain by including machine-interpretable definitions of basic concepts and relations among them. In recent years the development of ontologies has been expanding from academic areas to the desktops mainly because of the *Semantic Web* (). The ontologies on the Web range from large taxonomies categorising Web sites to categorisations of products and features for sale in a site (e.g. Amazon.com). The underlying objective of the *Semantic Web* is to provide a language for encoding knowledge on Web pages to make it understandable to electronic agents searching for information. Today, many organisations are building their own domain specific ontologies to share and enable reuse of common understanding of the structure of information among people or software agents.

In this sense, Schema.org (Ronallo, 2012) is a collaborative, community activity with a mission to create, maintain, and promote ontologies for structured data on the Internet and beyond. They are trying to define a general purpose vocabulary that covers entities, relationships between entities and actions, and can easily be extended through a well-documented extension model. Over 10 million sites and applications use Schema.org to markup their data for computer systems to provide richer experiences. Founded by Google, Microsoft, Yahoo and Yandex, Schema.org vocabularies are developed by an open community process.

When modelling agents that represent real world entities, it is needed to go through a compilation process about what attributes each agent should contain. Like in object-oriented programming (where the base class is Object and child classes go adding attributes and functions to this), real world element types also compound a family tree (properly ontology) that groups common attributes and further goes expanding child elements with their individual characteristics. Figure 3.3 shows the part of the ontology tree for financial products. The Schema.org community, already has done that big effort in identifying main real world entities, relations and which attributes should these have. Following their schema avoids having to repeat the identification process and contributes to using a common ontology already in place in a large number of applications and systems.

GWSAgents class hierarchy has been built according to the Schema.org ontology. Logically not all the schema has been implemented, only the sections related to humans (Schema.org Community, 2018b), physical structures (Schema.org Community, 2018c) and some organisations (Schema.org Community, 2018a). Figure 3.4 depicts part of the implemented class hierarchy in GeoWorldSim. As can be appreciated, for example instancing a BusStation will result in having all the attributes of GWSObject, GWSAgent, Place, CivicStructure, which conform the inheritance tree, and BusStation. If directly create one of these classes they will have by default an already created behaviour and skills.

One of the big contributions of the platform is that it is possible to construct any type of object starting from a Schema.org compliant file. Custom GWSAgent creation, can be achieved by two means, directly coding in C++ an agent or using the Schema.org ontology as high-level language. Instead of creating an own language (like some state-of-the-art platforms do), it was decided to adopt Schema.org ontology as high-level mechanism to build complex agents and behaviours.

To this end, GeoWorldSim provides an Object Factory mechanism for reading JavaScript Object Notation for Linked Data (JSON-LD) files (Consortium et al., 2014) and build GWSObjects from them. It is programmed using the Factory pattern (Gamma et al., 1993) (one of the most popular programming design patterns). Said pattern allows dynamically creating objects during runtime without exposing the creation logic to the user and through a common interface no matter the type of object to build.

As input for the factory, JSON-LD is a *Linked Data* (Bizer et al., 2009) format for creating computer readable inter-operable entities. The factory reads Schema.org ontology JSON-LD files to add *context* to the data and *coerce* values to be easier to process. Software using JSON-LD usually results in data structures that look like simple JavaScript Object Notation (JSON) (e.g. Section 3.2.1) but are in fact full *Linked Data* graphs. Inside the JSON-LD standard, properties that restrict or give more than just properties information are preceded by the `@` character. The most used constraining keywords are `@context`, to define the short-hand names or terms

that are used throughout the document; `@type`, to restrict an instance to an object; or `@id` to enforce the unique identifier of an element.

The GeoWorldSim Object Factory is key aspect for allowing distributed simulations that need to share agents across different computers. It registers Qt meta-objects that afterwards are matched to JSON-LD documents through the `@type` field. Given a JSON-LD description of an object, the Object Factory will read the `@type` value, go to its registered meta objects and, if said type exists, invoke Qt metaobject's `newInstance()` method to dynamically create a new entity.

The entities created will go through a deserialization process which firstly will search for the `@id` keyword. If exists, this value is set as the objects unique identifier since the entity can come from another parallel simulation and it is necessary that their identifiers coincide. If no `@id` is found, the simulator generates its own identification based on the *simulation identifier* + *type* + *number of object* (e.g. `SIM4-CAR1`, `SIM3-PERSON34`, `SIM3-BANK35`). Next, the rest of the keys of the JSON-LD document that do not start with the `@` character are parsed and introduced as meta-properties of the newly created entity. More complex objects, i.e. agents, require some other control properties (further detailed in Sections 3.3 and 3.4) which are transformed into not only properties but behaviour and capabilities.

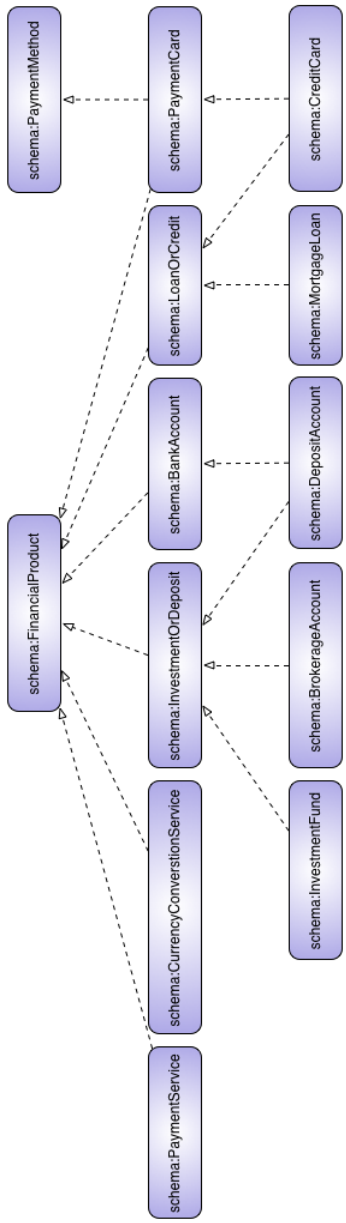


Figure 3.3: Financial products taxonomy. (Source: Schema.org)

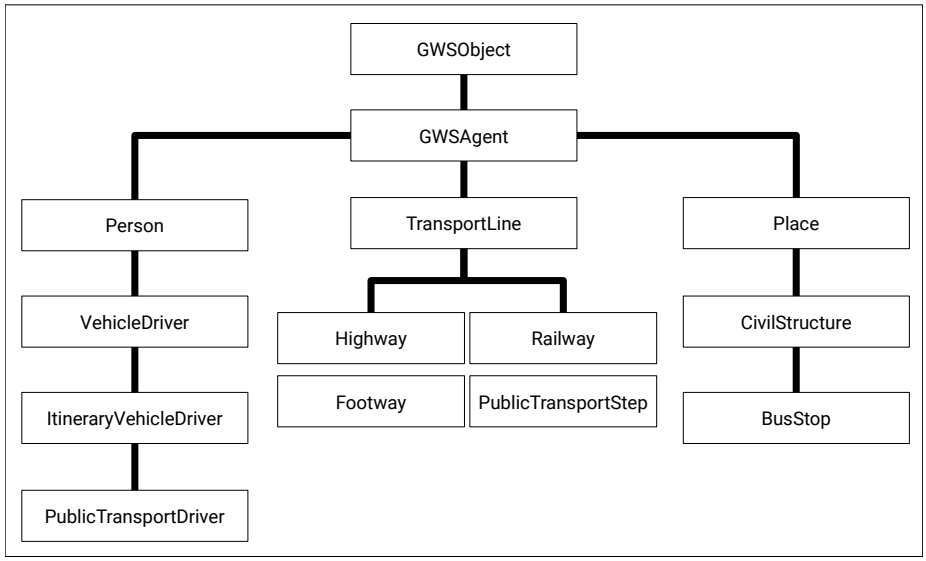


Figure 3.4: Part of GeoWorldSim agents taxonomy for transport modelling. (Source: own research)

```
{
    "@type" : "HumanAgent" ,
    "waste_amount" : 124.28 ,
    "@id" : "Citizen231" ,
    "geo" :
    {   "@type" : "GWSGeometry" ,
        "type" : "Point" ,
        "coordinates" : [ -2.84024 , 43.28139 ] } ,
    "@skills" : [
    {   "@type" : "ViewSkill" ,
        "view_agents_type" : "ContainerAgent" ,
        "view_geom" :
        {
            "@type" : "GWSGeometry" ,
            "type" : "Polygon" ,
            "coordinates" : [[ [-1, -1],[-1, 1],[1, 1],[1, -1],[-1, -1] ]]
        }
    } ,
    {   "@type" : "MoveThroughRouteSkill" ,
        "maxspeed" : 25
    } ] ,
    "@behaviours" : [
    {   "@type" : "MoveThroughRouteBehaviour" ,
        "@id" : "BH6" ,
        "duration" : 1000
    } ,
    {   "@type" : "FindClosestBehaviour" ,
        "@next" : "BH6" ,
        "duration" : 1000
    } ]
}
```

Code 3.2: Example of an agent creation from JSON-LD format. (Source: Own research)

3.3 Skills

The skill sets defined in GeoWorldSim are also a central point of the platform. Skills are the degree of competence that differentiates and empowers people to perform tasks and do so with agents and passive entities. The skills serve both to characterise entities in the MAS and are the way to perceive or propagate actions into the Environment and other entities.

Users building a Geosimulation will add skills to agents for them to be able to perform e.g. movements, calculations, geographic transformations, changes in others, etc. Taken to software, skills are the reusable code parts for agents to query the environment, process those perceptions, perform calculations and spread them back to others. Skills serve as reusable code pieces that stay with an agent during its entire life and glue between the agent's goal (modelled within the *Behaviour*) and the real code that performs such. So generally Skills consists of third party pre-written code, classes, procedures, scripts, configuration data and more.

Every skill needs to be contained within a context and behave under some restrictions. It is believed that the Environment needs to hold the responsibility of ensuring that these constraints are met and limit the capabilities of each agent type. So, although skills may change some agent properties (e.g. the geometry of an agent), the Environment may reverse or modify this changes (e.g. an agent without a flying skill setting its geometry to be several meters above the ground) to ensure real world constraints are met.

3.3.1 Underlying libraries

To code all the current skills into GeoWorldSim, this agglutinates several third party libraries. Using success proven external code avoids spending time and effort solving problems that others already did and allows to focus programming on the environment and agent's needs. Here is the list of used libraries:

Geometries: Developing a situated multi-agent system, geometries are a key part of the code. All the spatial representation and operations of GeoWorldSim has

been delegate in GEOS - open source geometry engine (Ramsey). This library offers an C++ implementations of all the simple features objects found in the OpenGIS *Simple Features for Structured Query Language (SQL)* specification (Consortium et al., 2003) and implementations of all the methods defined for those objects. Topological calculations are easy to visualize, but hard to implement generally. The GEOS algorithms are robust for all the spatial predicates (geometric comparisons which return true/false values) and also strong in the spatial operators (geometric functions which produce geometric results).

Projections: Geometries usually require coordinate transformations from one geodesic projection to another. For example, shaping geometries in latitude and longitude coordinates is not useful for metric distance calculations. This is where PROJ library (Evenden, 2005) has been used as it offers a generic software to transforms geospatial coordinates from one coordinate reference system (CRS) to another including cartographic projections as well as geodetic transformations.

Spatial index: GEOS library comes also with built-in Quadtree (Kothuri et al., 2002) spatial index for fast accessing geometries by location. A quadtree is a spatial index structure for efficient querying of 2D rectangles. All spatial objects can be represented by their envelopes (a rectangle) and the quadtree provides a primary filter for range rectangle queries. Figure 3.5 shows a descriptive image of a quadtree. This starts creating a single rectangle where the first few geometries are all together and while adding more it will explode those rectangles whose number of geometries increases into sub-rectangles, automatically expanding to accommodate any extent of data-set.

Graphs: Graphs creation in the Network Environment is also achieved through GEOS and its Plannar Graph classes. They represents a directed graph which is embeddable in a planar surface. These classes serve as a framework for building planar graphs for specific algorithms and allows controlling the types of graph components (edges and nodes) which can be added to the graph.

Voronoi: Voronoi diagrams and Delaunay triangulations (Fortune, 1995) are dual processes for partitioning of a plane into regions based on distance to points

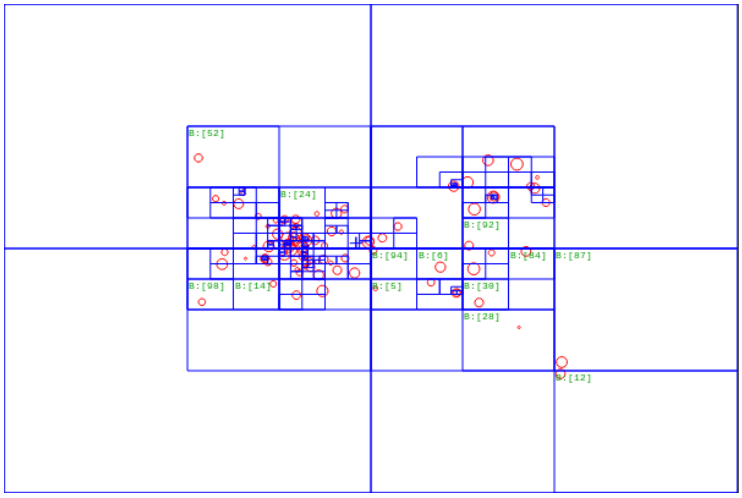


Figure 3.5: Representation of a quadtree for geometry indexation. (Source: Lucidation contributor to Wikipedia)

spread along the plane. Within the partitioning, each point (called seed, site, or generator) will be contained in a polygon that delimits all the region consisting of all points closer to that seed than to any other. These regions are called Voronoi cells. Figure 3.6 shows an example of a Voronoi diagram where each region has a different colour.

Dijkstra: Many of the operations an agent must perform entail finding the location and route to resources. Shortest path problems involve modelling the environment as a network (graph edges and nodes) with the costs of travelling per each edge. Dijkstra's algorithm (Dijkstra, 1959) finds shortest paths from source node to all nodes in the given graph and calculates the cost of performing such route. The shortest-path calculations have been delegated to Lemon Graph library (Dezső et al., 2011) a generic open source C++ library providing easy-to-use and efficient implementations of graph and network algorithms and related data structures. From a GEOS graph, GeoWorldSim generates a Lemon graph

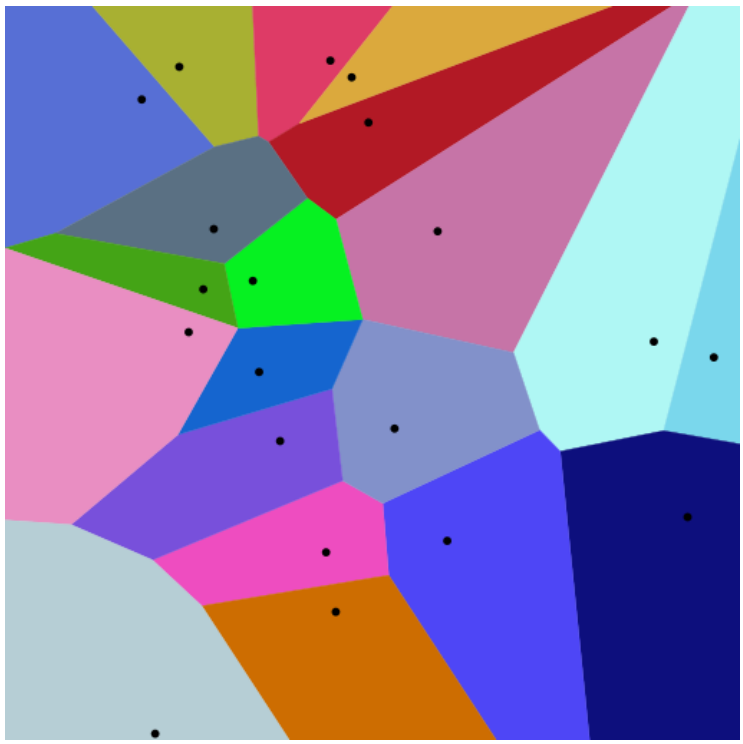


Figure 3.6: Voronoi diagram where seeds are contained in their region. (Source: Balu Ertl contributor to Wikipedia)

for calculating shortest routes and travelling salesman problems (Dezső et al., 2018).

Service area: When using shortest path algorithms, another recurrent technique is the so called Service area calculation. Given a source node, a network service area is a region that encompasses all accessible nodes, that is that are within a specified impedance. This performs shortest-paths on a network to expand through all the reachable network edges until reaching the distance from the source. It is a powerful mechanism to calculate the spread of an event or the

range of operations. Figure 3.7 displays the reachable area within 30, 40 and 50 minutes from a point.

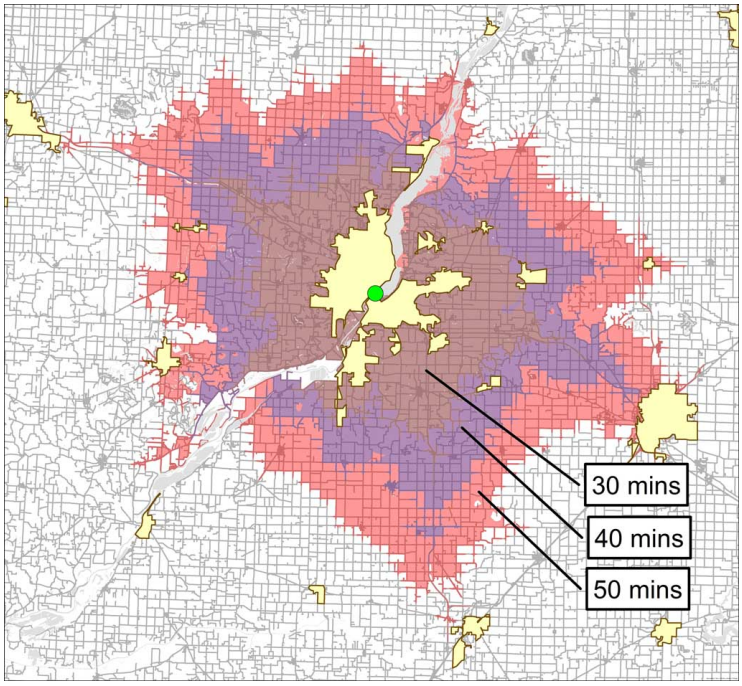


Figure 3.7: Service area accessible from a point. (Source: ESRI)

Neural networks: Neural networks (Nissen, 2005) are a series of algorithms that attempt to identify underlying relationships in a set of data by using a process that mimics the way the human brain operates. They have the ability to adapt to changing input so the network produces the best possible result without the need to redesign the output criteria. To this end, LibFann Fast Artificial Neural Network Library (FANN) (Nissen et al., 2006) is a free open source neural network library, which implements multi-layer artificial neural networks. It has been integrated into GeoWorldSim through its C support for both fully connected and sparsely connected networks.

Fuzzy logic: Dealing with human behaviour it is necessary to model complex decision-making based on different weightings of facts. To this end, Fuzzy Logic (Berkan and Trubatch, 1997) has proven to achieve high success in the emulation of individual preferences (Tuzkaya and Önüt, 2008) enabling a wide level of parametrisation. Fuzzylite engine (Rada-Vilela, 2018) builds a discrete choice model through a simple natural language rule-based approach. Classical logic only allows conclusions to be True or False. In contrast, Fuzzy Logic defines a set of rules that map numeric data into linguistic terms and create fuzzy sets indicating the extent to which each term is part of. That is, a variable can have several values (called terms) that can be overlapped and shaped. Thereby, it is possible to model how each feature sums or decreases the decision of an agent.

Support vector machines: Super Vector Machiness (SVMs) are supervised learning algorithms well-established in the research community and are integrated in GeoWorldSim through the LibSVM (Chang and Lin, 2004) library. SVM provide classification and complex calculation skills. These algorithms are especially good at data classification (i.e., identifying which category a new observation belongs to based on a training set of data containing known observations) or for extracting linear functions from discrete values (i.e., having some point measurements, create the linear function to generate any measurement).

3.4 Behaviour

As said, all agents have a certain behaviour (which may be empty or complex) that orders and orchestrates the use of skills. Agent platforms tend to iterate across all agents' behaviours by means of the `tick()` method: a function called periodically. Using the `tick()` method, state-of-the-art agent platforms use different mechanisms to model the operations an agent must perform. In this line, state-of-the-art Jade platform implements Behaviour classes or blocks (Bellifemine et al., 2018) that can be chained to define an agent's tasks. A quick look through these, shows up to five extendable primary classes:

SimpleBehaviour: Task to be executed once.

CyclicBehaviour: Task to be executed repeatedly.
OneShotBehaviour: Task executed once after a timeout.
ParallelBehaviour: Task that executes a set of child behaviours in parallel.
SequentialBehaviour: Task that executes a set of child behaviours in sequence.

At this point, GeoWorldSim's behaviour model follows the well established hierarchy of *Activity* and *Action* (Kamara-Esteban et al., 2017a). *Actions* are short-timed events *executed* by an agent, such as grabbing a cup of tea or opening a door. *Activities* are seen as *sequences of actions* in a time frame ranging from minutes to hours, such as preparing breakfast, making coffee or brushing teeth. The difference between an *Action* and an *Activity* is not only the time lapse, but also the inherent higher semantic level due to the existence of more complex interactions between objects and people.

The agent's *Behaviour* is composed of a set of ordered *Activities* called the `TODO list`. *Activities* are indivisible pieces of behaviour that that define a series of *Actions* that need to be done before moving on to the next *Activity*. This inner *Actions* do not need to follow any established order to fulfil their parent *Activity* and so can be executed in parallel.

This simple scheme allows to combine the execution of behaviours both in sequence and in parallel. On the one hand, if a pure sequential conduct wants to be modelled, then only *Activities* need to be loaded to the agent and this will finish one after the other. On the other hand, if a complete disordered and parallel execution of pieces of behaviour is wanted, then a single *Activity* with as many *Actions* as wanted can be loaded into the agent.

Shaping this into code, GeoWorldSim implements a GWSBehaviour class which does the function of both *Activity* and *Action*. In this way a GWSBehaviour representing an *Activity* can have a list of sub behaviours which would represent the *Actions*. Every GWSBehaviour class contains the following methods and attributes (depicted in Figure 3.8):

`behave()`: Virtual method to perform that *Behaviour's* specific code. This method can access the behaving agent to alter its properties and skills. Through these

last ones the behaviour can query the environment or provoke changes in others.

`duration`: Duration of the *Activity* or *Action*. When executing this, the agent's execution will not be called again until said time has past. If more than one *Action* is executed in parallel, the highest duration is taken.

`sub_behaviours`: A GWSBehaviour serves for modelling both *Activities* and *Actions*. To this difference these, a GWSBehaviour can have child GWSBehaviours to shape the *Actions* belonging to an *Activity*.

`finished()`: Virtual method for the *Behaviour* to tell if it has finished. An agent will search the *Behaviours* whose `finished()` method returns `false` to perform them.

`finish_condition`: A parent GWSBehaviour's completion, will depend on whether its children have been completed. However there can exist *Activities* that do not need all their *Actions* to be performed to finish. To this end, each parent GWSBehaviour contains a `finish_condition` number that states how many of its sub_behaviours need to be finished to set the parent as completed.

`next`: A pointer to the next *Behaviour* that should be executed after this. It is used both for defining sequential *Activities* or preconditions inside parallel *Actions*.

What Jade platform implements through five Behaviour classes (Bellifemine et al., 2018), GeoWorldSim offers with a single GWSBehaviour class, the agent's `internal_datetime` and the parent-child relationship. Further explained; on the one hand, Jade's Cyclic behaviours can be achieved by setting the agent's `internal_datetime` and the behaviour's `finished` attribute to adjust the cyclic or one-shot finished calling. On the other, Parallel behaviours are achieved by adding children sub-behaviours to a GWSBehaviour and Sequential behaviours through the `next` behaviour linking.

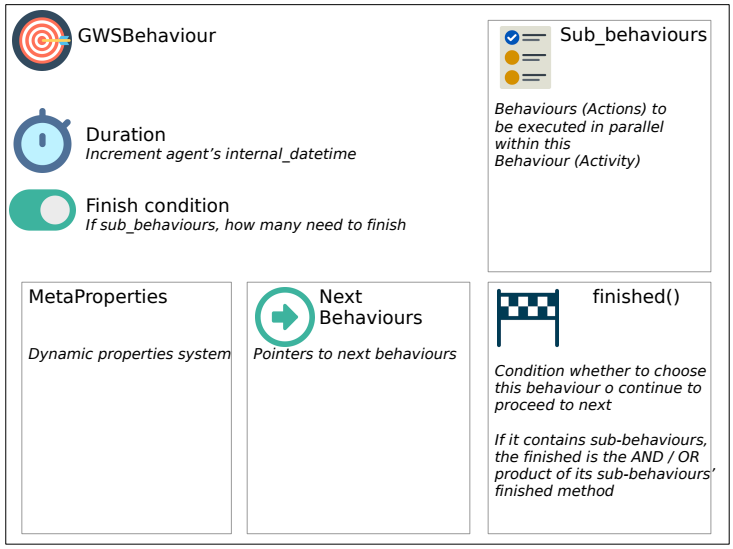


Figure 3.8: GeoWorldSim's behaviour block. (Source: Own research.)

Behaviour executing

As will be further explained in Section 3.6.7 agents contain two `busy` and `internal_datetime` properties essential to manage their behaviour:

`busy`: A number that keeps track of how many *Behaviours* is the agent performing. Avoids calling the next tick of the agent if any *Behaviour* is still pending, that is, if the `busy` attribute is greater than 0.

`internal_datetime`: Datetime which tells, despite the simulations' real time, how much the agent has advanced in time. This `internal_datetime` sets when will the agent be *ticked* once not `busy` anymore. When executing a GWSBehavior, its duration will be added to the agent's `internal_datetime`. As mentioned, if *Actions* are run in parallel, the highest duration of them will be added.

Whenever an agent is *ticked*, it will iterate the `TODO list` checking each GWSBehaviour's `finished()` method to find the next unfinished one. The *Behaviour*

algorithm can be understood as a *Decision Tree* with backtracking and pruning (Quinlan, 1986) (as shown in Figure 3.9) that iterates *Activities* and *Actions* until finding the non-finished ones to perform them. Every agent `tick` the algorithm will start its search from the first *Activity* of the *Behaviour* deepening in the tree while it finds finished activities. As mentioned, an *Activity* is a block with many *Actions* inside. According to the the nature of the found behaviour, two different executions can happen:

Non children containing GWSBehaviour: `behave()` method will be invoked to perform the *Activity* that firstly increments the agent's `busy` attribute and then executes its code to query, process or modify some attributes from the agent or Environment. Once the behaviour has been executed (may have ended or not), the agent's `busy` attribute will be decremented back to 0.

Children containing GWSBehaviour: `behave()` method will invoke all its unfinished children's `behave()` method in parallel. Each of these, will increment and decrement the `busy` attribute to keep track of when all parallel *Actions* have finished.

Figure 3.9 exemplifies a *Behaviour* that starts at `A0`. Green represents the finished GWSBehaviours consulted by the algorithm (e.g `A0`, `A1`, `A2`) and red unfinished ones (e.g. `A4.2`, `A4.1.2`, `A5.2`, `A6`). Notice how the decision tree algorithm together with the parent-child GWSBehaviours would act in the depicted image:

- Asks if `A3` has finished. This checks that all its children have finished and returns true, so the algorithm can head to the followings (`A5` and `A6`).
- Asks if `A5` has finished, which will notice that one of its children has but it does not reach the `finish_condition = 2`. As it has not been finished, the algorithm will mark `A5` for execution, and this execution will result in calling `A5.1`'s `behave()` method.
- Asks if `A6` has finished and since it has not, marks it for execution.
- Asks if `A2` has finished. It will notice that one of its children has and reaches the `finish_condition = 1` so no more children need to be queried and the algorithm can head to the followings.

- Asks if `A4` has finished, which will ask its children. Only one child has finished therefore the `A4`'s `finished()` method will return `false` and be marked for execution.

So, once collected the *Behaviours* that need to be executed, the agent will call `behave()` of:

- `A5` and this will invoke `A5.1`.
- `A6` which will perform its code.
- `A4` and this will invoke `A4.2` and `A4.1.2`.

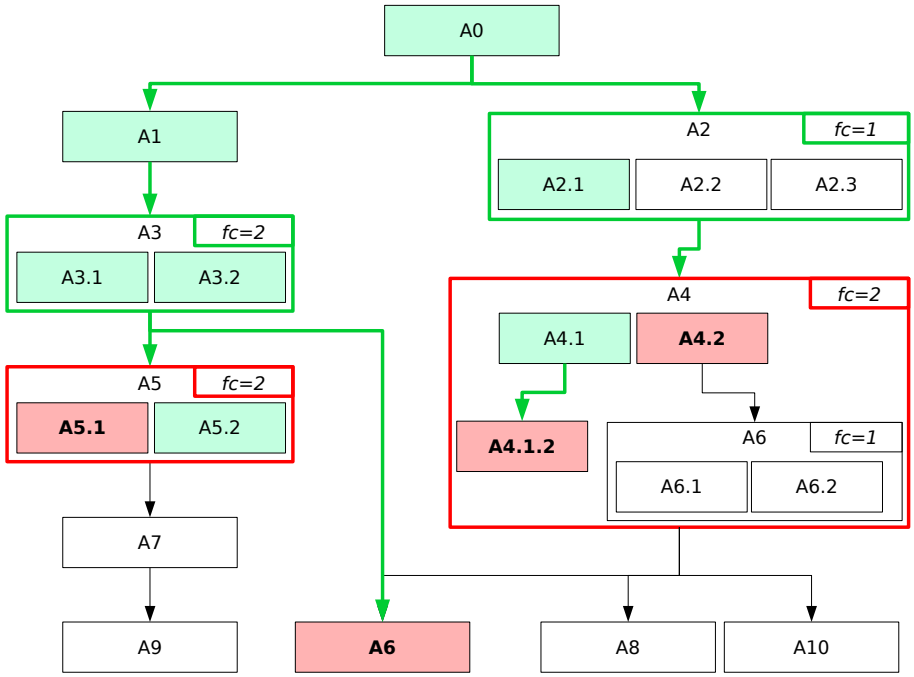


Figure 3.9: An example of a *TODO List* composed of *Activities* and *Actions* inside them. The green ones have finished and therefore the Behaviour algorithm iterates through the `next` arrows for finding the unfinished ones. (Source: own research)

Running a GWSBehaviour's `behave()` method does not ensure that the goal has been reached. A `behave()` could be just to move a couple of metres closer to a certain position and, therefore, must be *ticked* until reaching it. The achievement of the behaviour's goal is certified by the `finished()` method that checks if the goal is met and, therefore, it is the agent's duty to check at every *tick* if it should perform an *Activity* not completed.

3.5 Complete GWSAgent

Following the Figure 3.2 and once loaded a set of Skills and Behaviours, an example of a *Citizen* driving to a destination would be completed as depicted in Figure 3.10. Its execution would go through the following steps:

1. First the `Calculate walking route to vehicle` behaviour would require its `Calculate Route Skill` to extract the route the agent should follow until reaching the vehicle.
2. It would follow said route using its `Move Skill` until the `Walk to vehicle` behaviour finishes.
3. Then the agent would proceed to the `Calculate driving route to destination` behaviour to call again the `Calculate Route Skill` but this time by passing its final destination and using the vehicle.
4. It would loop executing in parallel the `View Cars` and `Drive` behaviours. The `View Cars` behaviour, uses the `View Skill` to look at the preceding vehicle in the road and decide the desired speed; and the `Drive Vehicle` behaviour, alters the vehicle's speed for it to use its own `Move Skill` that will transport the agent.
5. When these last two behaviours return the `finished()` method as true, the agent will have reached its final destination.

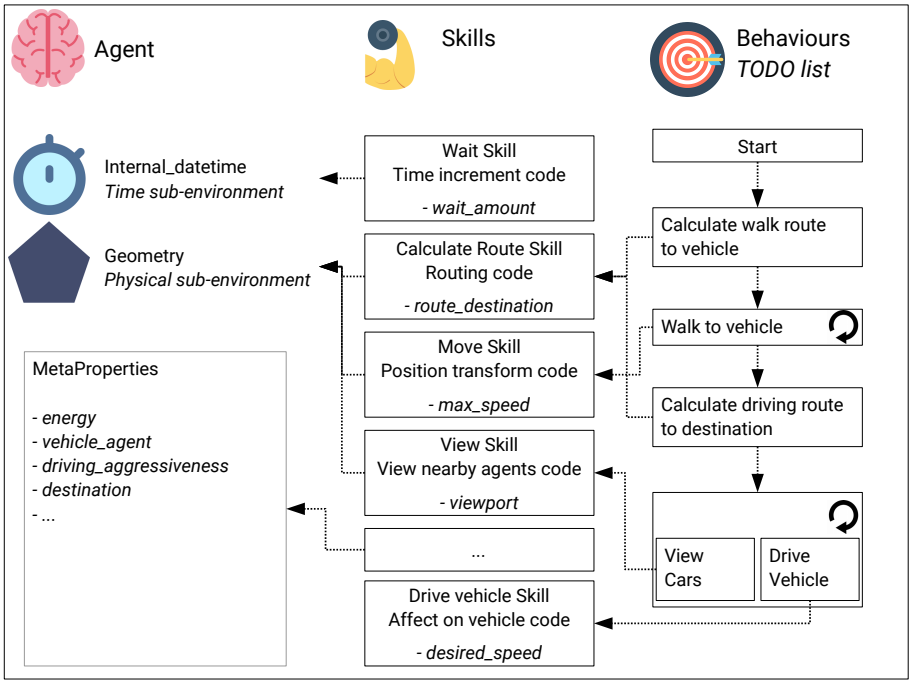


Figure 3.10: Full inside elements that compound a GWSAgent driving to a destination. (Source: own research)

3.6 Environment

The agent environment is the world where all active agents and passive entities live together and interact. An appropriate environment definition must bring together real world scenarios and applications by addressing key issues as time management, agent registration and real world constraints assurance. In real life, the world is omnipresent and we access it instinctively and continuously through our senses. Same happens in GeoWorldSim where the world is modelled through the GWSEnvironment class. This virtual omnipresent world is programmed by a *Singleton* design pattern (Gamma et al., 1993), concept that fits perfectly into how an agent environment should be translated into software. *Singletons* are accessible from any part of the

code, without needing a reference to it, through a static method and restrict the instantiation of it to a single object. Therefore, any agent gets an access point to the GWSEnvironment or any of its sub-environments to interact and provoke changes.

There are two big contributions on how the environment has been coded. Firstly, GWSEnvironment is an agent itself willing to have a behaviour like any other through its `tick()` function. Most MAS fail in treating the environment as a first class and proactive entity with its own behaviour (Weyns et al., 2007b). And secondly, within GeoWorldSim the environment class contains a list of sub-environments that serve to model the collective global behaviours in which the world decomposes.

When bringing to life or destroying an agent from the main environment, this will cascade the agent to its child sub-environments so these can filter, modify or add additional properties inherent to the type of the real world aspect they represent (e.g. an agent being registered in the physical sub-environment will get attached a geometry to it). Later when *ticking* the global GWSEnvironment (remember it is an agent), this will call its child sublayers' `tick()` method.

Through these, the proposed powerful design allows extending the environment by means of creating new sub-environment layers to model collective behaviours and external tools (e.g. institution groups, routing networks, physical models, rain models, weather models, power grid engines). Currently some sub-environments have already been programmed, and they are further detailed bellow.

The most important of the inside built-in layers, is the Execution sub-environment in charge of *ticking* agents and with a new approach for running distributed Geosimulations. The GWSEnvironment publishes its state through the external communication module Section 4.5 and likewise, it receives the status of other environments. While the current state-of-the-art platforms need to explicitly deploy mechanisms and messaging for time federating distributed simulations, the GWSEnvironment can have inside external environments masked as normal agents. This masked agents, will have no behaviour and their inner time will be updated from the outside. Combining the methodology bellow described in Section 3.6.7 of the Execution Environment (which waits for delayed slow agents) together with hiding external environments as agents achieves a transparent synchronisation of distributed executions. Fig-

ure 3.16 shows in a more explanatory way the process of time synchronisation among distributed simulations.

3.6.1 **Agent sub-environment**

Agents directory that organises and enables the search of agents by type or unique identifier. The environment requires a full list of all its living beings for fast searching and retrieving. This Agent Environment is the layer where agents are registered and decomposed into their ontology tree, that is, agent indexes are structured from a generic index with all of them to the most specific ones where only agents of a certain type can be found. Figure 3.11 shows these ontology indexes that store the agents and allow retrieving all that match a certain type.

To decompose an object into all its inheritance tree, the Agent Environment uses the GWSObject meta-objects. Qt's object-model allows climbing up from one meta-object to its super class and so on until receiving a null meta-object which will mean having reached the base class. Section 3.6.1 illustrates the iteration since it is relevant for many sub-environments of GeoWorldSim.

```
1 QStringList GWSObject::getInheritanceTree() const{
2     QStringList l;
3     const QMetaObject* obj = this->metaObject();
4     while( obj ){
5         l.append( obj->className() );
6         obj = obj->superClass();
7     }
8     return l;
9 }
```

Code 3.3: Qt's MetaObject iteration for climbing up parent classes. (Source: own research)

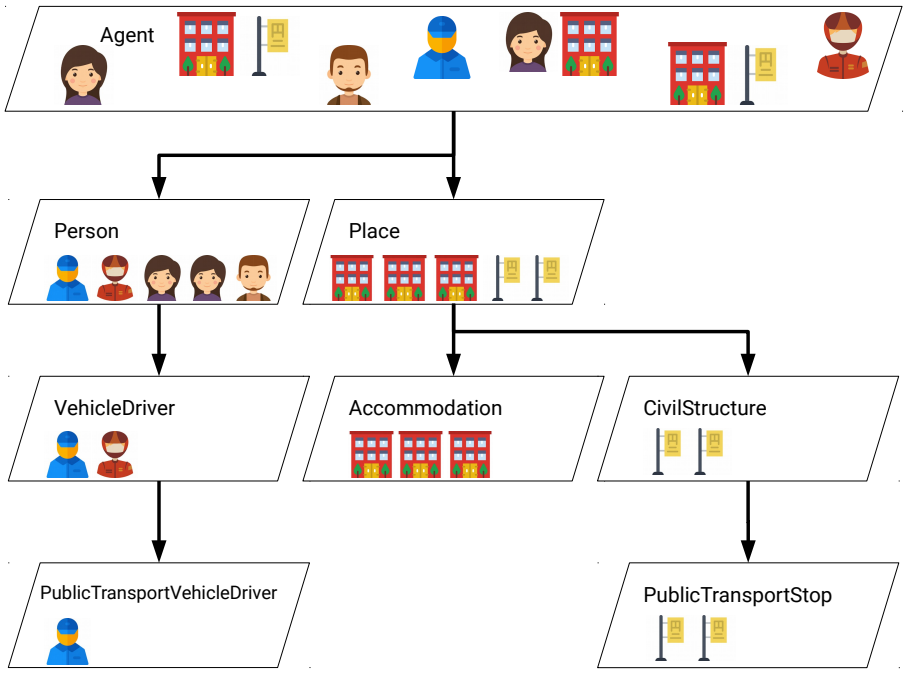


Figure 3.11: Agent environment storage to index by type. (Source: own research)

3.6.2 Time sub-environment

Time is the progression of events and this is what this sub-environment is dedicated to and what drives the simulation forward. The Time Environment is a simple class that saves two key properties:

Simulation time: When created, the Time sub-environment saves a creation timestamp and the simulation speed. Using both it is possible globally access the current date and time of the simulation by multiplying the time spent in the real world by the desired speed of the simulation.

Agents' internal time: As previously introduced, agents are not always in sync with the simulation time (e.g. because of heavy processes, waiting for an external

result, etc.). The time sub-environment contains the `internal_datetime` of all registered agents and use it to increment said time according to their progress.

3.6.3 Physical sub-environment

Physical layout where the agents live composed of a three dimensional terrain model that delimits the study area and serves as a spatial index to place all existing agents. Every agent needs an attached geometry representing its physical properties (e.g. a pipeline modelled as a LineString, an accommodation as a Polygon, an entrance to a building as a Point, etc.). GeoWorldSim contains a geometry engine built on top of the third party GEOS library. When adding agents to the Environment, this physical layer will ensure the agent fits inside the environment bounds and create a cascaded spatial storage (just the same as with the Agent Environment but for spatial searches).

The updates of the geometries of agents, have to necessarily pass through this component. It will verify that no agent is making trying to move or fly if not allowed.

3.6.4 Network sub-environment

When talking about urban developments it is inevitable not to think the networks. Whether they are transport, sanitation, electricity, waste collection, etc. Urban developments are composed of a large set of networks. That is why GeoWorldSim's environment had to have a layer devoted to networks for registering all real world entities that can be shaped as a graph where elements are connected between them. To this end, two abstract classes have been created GWSGraphNode and GWSGraphEdge for agents to extend them: GWSGraphNode contains a list of all incoming and outgoing edges whilst GWSGraphEdge requires two *from* and *to* GWSGraphNodes. For example, fig. 3.12 depicts the class hierarchy tree for transport networks inside GeoWorldSim. A TransportLine would need two TransportNodes to be created since it represents a simple Highway and a PublicTransportSegment would need two PublicTransportStops because it represents an available route between two stops.

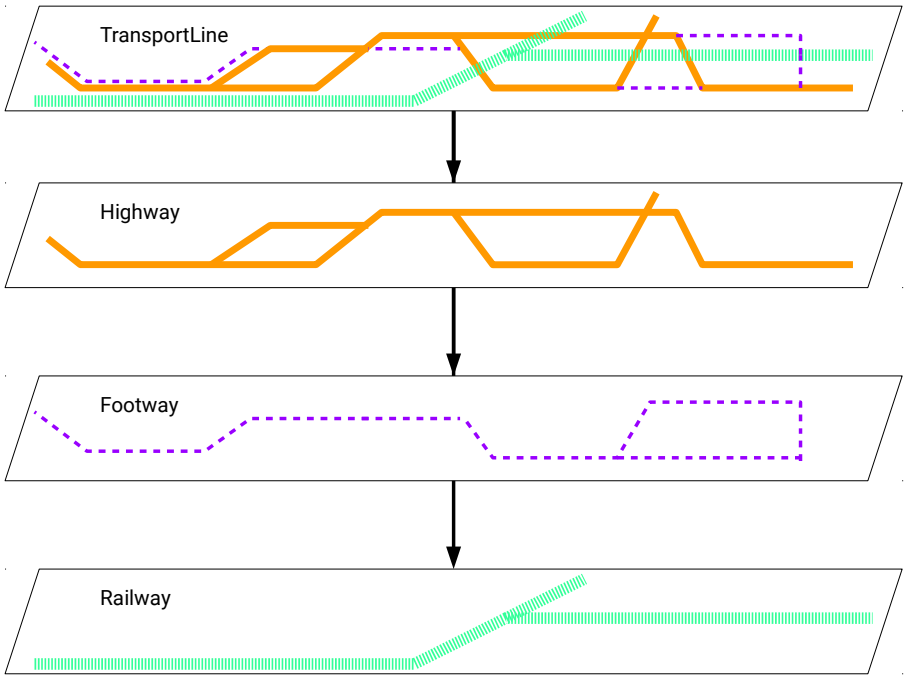


Figure 3.12: Transport network Graphs by extending GWSGraphNode and GWSGraphEdge. (Source: own research)

Joining both components several GWSGraphs are built inside the Network Environment. Whenever an agent is registered in the environment, it is passed to the Network Environment who checks if the agent can be cast to a graph edge or node. If so, the function iterates all the inheritance tree of the agent creating a GWSGraph and inserting the agent in it. Through this process, several graphs are created, ranging from the most generic agent types to the most specialised. The Network Environment, exposes spatial search methods for easy network building and reuse of elements. Using these graphs created, GeoWorldSim

3.6.5 Social sub-environment

Many Geosimulations require modelling principles or norms that govern agents' behaviour, understanding that these are based on implicit or explicit agreements (e.g. institutions, family, affinity). This sub-environment focuses on modelling social groups and hierarchies between agents. To this end, it creates a set of social graphs (Ugander et al., 2011) to represent how each pair of agents are socially connected (e.g. hierarchy in a company, friendship, family tree). Using these, the Social sub-environment allows to check if a certain relation exists between two agents, for example if `AgentA` is *FRIEND* of `AgentB` or `AgentA` is *EMPLOYEE* of `AgentB`.

3.6.6 Messaging sub-environment

Geosimulations recreate worlds where agents compete or cooperate with each other and messaging plays a vital role. Inside GeoWorldSim, this sub-environment enables an active listener to orchestrate messages emerged within and outer distributed Geosimulations:

Messages emerged within the simulation: Agents living together in the same Environment propagate messages to other. A set of messaging Skills will emit said messages to this sub-environment who will connect them to other agents' slots. Using the Signal-slot mechanism, ensures the asynchrony of messaging and that the processing of emission and reception is done both in their respective agents' threads.

Messages coming from the outside: Distributed simulations can be a source for messages that need to provoke changes or are expecting to receive a reply. The Messaging sub-environment opens as many active listeners as required using Websockets (and the module in Section 4.5) that forward messages to their recipient agent's slot. This messages can be either generated from distributed simulations or created on the fly by third party applications and sensors (with no simulation infrastructure). The messaging layer will find no difference as long as these are well formatted.

In carried out simulations agents only compete with each other and still no negotiation mechanisms have been implemented. Therefore, messaging is currently limited only to the alteration of attributes of one agent by another (e.g. a driver changing the speed of a car, commuters hoping on a bus, turning on and off a household appliance). This dissertation has very present that to extend communication and interoperability among different agents, the FIPA ACL (Fipa, 2002) messaging protocol is the universal mechanism in the MAS world. Inside FIPA's communicative acts, there are already message types for operations such as making proposals, agreeing or rejecting between agents. It is intended to endow GeoWorldSim with FIPA messaging capabilities in the short run. To this end, it is expected to use existing communication library implemented in C++ (Roehr, 2018) which includes a message parser and a message generator for bit-efficient FIPA ACL messages.

3.6.7 **Execution sub-environment**

This logical layer for time management and asynchronism contains all the active agents that are alive and performing a certain task. Section 3.6.7 depicts the typical shape of a simulation loop in MAS. The loop ensures that all agents are synchronised and given an opportunity to behave at each simulation step. All GWSAgents have also been designed with a `tick()` function which serves as wrapper for calling the `behave()` method that runs its behaviour. This `behave()` is an abstract method enabling new agent types to implement their own specific function to emulate the real world entity, so that calling all agents's `tick()` results in different behaviours.

Ticking all the agents at each simulation step, like in NetLogo or Gama platforms, means reducing the agents' reasoning to the same unit of time and losing any asynchrony. In order to improve this, firstly the centralised control has been discarded in favour of giving agents autonomy to `tick()` themselves whenever wanted, as event-driven platforms do. Taking into consideration that the objective is to run Geosimulations with thousands of agents in computers that rarely have more than 50 processors, it is unavoidable to go beyond software and start analysing the hardware layer. Only a few (equal to the number of processors) tasks can be really executed in

```
int ticks = 0;

// Simulation loop
while( !finished ){
    for(int i = 0 ; i < agents.size() ; i++){
        agents[ i ].tick();
    }
    ticks++;
}
```

Code 3.4: Traditional structure of agent steps execution in MAS. (Source: own research)

parallel by the computer and here comes into play the way in which the operating system implements multitasking and thread scheduling. Computers run several processes in parallel and within a process several threads. When running a process, it always starts with a single thread (usually called main thread) and during its execution may decide to create more of them (called worker threads). Operative systems use different scheduling algorithms that attach a priority to threads alternating the execution of these so that, a higher priority has more potential to receive resources from the operative system. This means that, even if several agents can invoke a `tick()` to be executed at the same time, the operating system will go sequentially *turning on and off* their execution.

The work here had to face how to make GeoWorldSim agents to run in a fast, asynchronous and concurrent way and this is where the need for a Execution Environment layer emerged as a mechanism to *control* (not to synchronise) the asynchronous execution of the agents.

Qt provides easy thread management by means of four low and high level techniques (Qt Company, 2018b) for creating concurrent pieces of code. Some of these are oriented for launching short life-cycle worker threads to delegate single tasks and are not what is needed. Agents must be hosted in their own long-living thread that can perform different tasks upon requests and react to new data and stimuli. To this end, low level QThread class is the foundation of all thread handling in Qt, is the most flexible way of obtaining full control of concurrency and each QThread has

its own *event-loop* for signals/slots triggering. Recommended use is to instantiate a new QThread instance and then go through the `moveToThread()` method to transfer a parent QObject instance (with all its child objects) to the newly created thread. When moving objects to threads, pointers and data are shared across all QThreads, i.e. no separate memory spaces are created. Therefore, having concurrent agents accessing shared data in a disorderly way, it is hard to ensure that all remains consistent. Problems are often hard to find because they only show up occasionally or on specific hardware configurations.

Per newly created QThread, Qt creates an *event-loop*. These are used to queue and schedule slots to be invoked immediately or after a certain time. It is not necessary to queue them in order of execution, the user can schedule them in a disorderly way and the *event-loop* will order and trigger the functions as scheduled. It is important to remark that there is only a single *event-loop* per thread, therefore if two slots are scheduled to be triggered at the same time, the first will be firstly executed and once finished, the second. Due to this, for a correct orchestration of the ticks, threads and agents, three approaches emerged:

Single worker thread: As mentioned, each program has only one thread when it is started. The usual procedure in Qt is leaving the main thread for user input catching and creating secondary worker threads to send these inputs' processing. It is fair to think that the most equitable way for active agents to receive the same thread priority and opportunities is keeping them all in the same worker thread. In this approach a single worker thread is created to move all the active agents to. Active agents' `tick()` function is a slot, and in this approach each agent used a timer that asynchronously triggered these slots.
Imagine an agent that emulates a train driver performing its itinerary. Inside the `tick()` function, the first task would be driving to the next train station until arriving there. Once at the station, if the driver had to obligatorily wait 15 seconds before departing, it would create a timer to queue for 15 seconds in the event loop the call to the `tick()` function again. This avoids busy-waiting with the agent in a loop asking whether it is time for calling `tick()` again or sleeping a thread (that may be shared and needed to be awake by another

resources). Doing so, Qt's *event-loop* works as an dispatcher in which ticks are queued and fired with no need to to implement a new execution controller. However, even though agents receive an equal allocation of resources, this technique does not take advantage of multitasking since there only exists one single event loop executing all agents' behaviours sequentially.

As many threads as agents: The opposite approach is creating and assigning one thread per agent for each to have its own event loop where to queue the ticks' executions. In this case, the allocation of resources to agents is delegated to the operating system which would assign threads to hardware cores using its scheduling algorithms. Because there will probably be more agents than cores in the computer, this alternative makes full use of multitasking and runs the ticks completely in parallel. However, concurrency is achieved by repeatedly pausing a thread, saving its memory state and reloading another thread's saved memory into the core; known as *context switch*. Context switches are usually computationally expensive since pausing a thread and awaking another requires a certain amount of time for doing the administration (e.g. saving and loading registers and memory maps, updating various tables and lists, etc.). Besides, Qt's `moveToThread()` function is also time consuming making no sense to move simple passive entities that have no active role in the simulation. These two issues, affect the performance and speed of a simulation resulting in agents receiving not balanced processor time. If some agents are forced to wait by the operative system while other receive more computation time, all entities in the simulation will go desynchronised in date and time.
Imagine the simulation of the evacuation of a building, where people run to the emergency exits. Having agents that can execute more ticks than others being delayed by the operative system would suppose to have people for whom time passes at a speed different from that of others.

Mixed strategy: As always, mixing strategies brings the advantages of both approaches. The GeoWorldSim Execution environment has been modelled as a layer containing only the agents that requested to perform an active role in the simulation. This layer is a singleton provinding two `runAgent()` and `stopAgent()`

methods which will give agents capacity to invoke ticks and concurrency. When the execution layer is firstly created, it also creates a Parallelism Controller, utility to manage multi-threading, that queries the number of processor cores α in the system, both real and logical, to create new $\alpha - 1$ QThreads. New threads plus the one already existing in the application sum the number of cores in the system and therefore the maximum number of operations that can actually be executed in parallel. Whenever an agent is passed to `runAgent()` function, the execution environment relies on the Parallelism Controller to randomly assign one of the newly created threads to the agent and register it in the execution layer. Now agents living in the same QThread will share an *event-loop* for triggering their `tick()` functions. The main thread is kept for the main application necessities and the passive entities (as they do not need to actively run any `tick()`).

In spite of using a mixed strategy to distribute all the agents across the computing capacity, it is not possible to know in advance the load each agent will need. Recall that the agents are scheduling the `tick()` calls in their *event-loop* with no centralised control. This generates situations where agents with high computational needs block their *event-loop* while other loops dispatch lighter tasks faster. Even more, GeoWorldSim is designed to execute simulations distributedly across several computers and these may have different hardware configurations. Agents within faster computers or none blocked *event-loops* will go faster in time just like occurred with the *as many threads as agents* approach. The difference is that in this case the platform is not delegating thread scheduling to the operative system and it is possible to implement at code level mechanisms for managing desynchronisation.

A new multi-agent execution management technique is proposed supported by a new Execution Environment. This new logical layer, sets up a moving time-window for asynchronously executing agent behaviour ensuring that lagging agents are not left behind. It has been designed and implemented, creating a new sub-environment inside the global Environment and which will attach two `internal_datetime` and `busy` properties to agents. In this novel technique, instead of explicitly queuing the next `tick()` inside the *event-loop* to be triggered after several seconds, agents

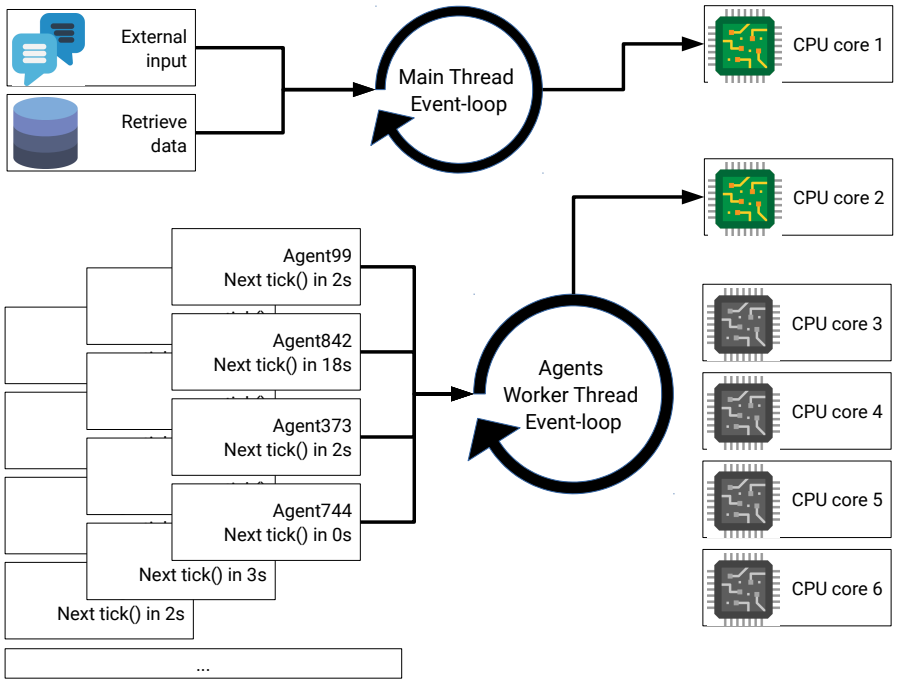


Figure 3.13: Worker thread approach for scheduling agent ticks. (Source: own research)

set a `internal_datetime` property indicating when they need their next operation to be triggered. It is responsibility of the Execution Environment to serve as link between the *event-loops*, agents and the Time Environment synchronising the moment all are simulating.

Whenever an agent *wakes up*, is run to execute its behaviour, it is registered in the Execution Environment layer, set its `internal_datetime` property to *now* and sent to a randomly selected worker thread and *event-loop*. The Execution Environment has a *tick()* method invoked periodically that will follow the procedure described in Algorithm tick. The sliding time-window progresses over the time but can be delayed if agents are late because of a heavy behaviour. All agents whose `internal_datetime` fall inside the time-window, will have their scheduled time

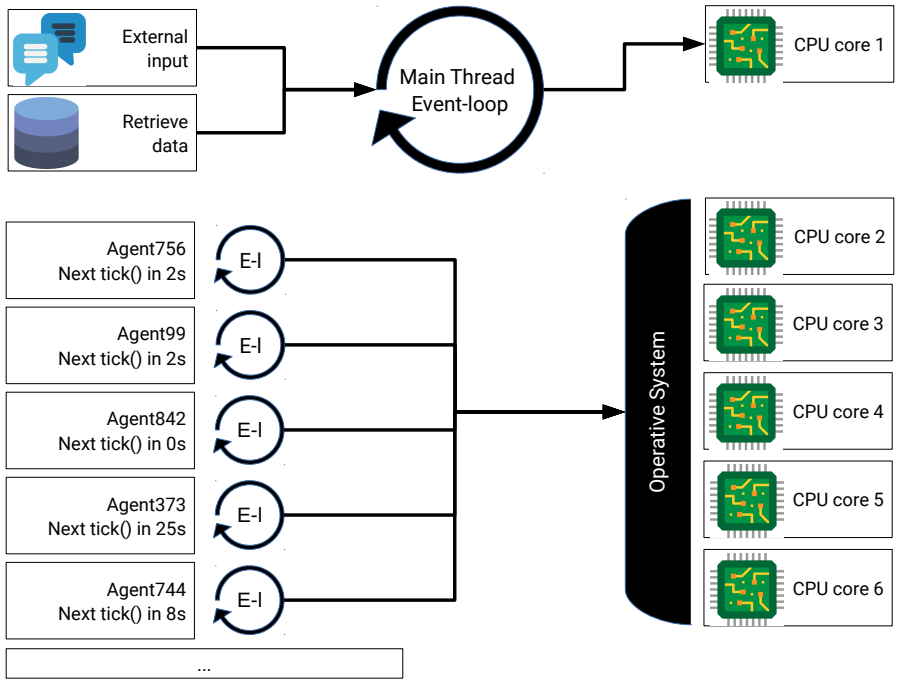


Figure 3.14: As many threads as agents approach for scheduling agent ticks. (Source: own research)

recalculated and, now yes, the `tick()` queued in the *event-loop*. All agent ticks within the time-window will be executed before calling again the next `tick()` iteration of the Execution Environment. This is one of the biggest innovations contained in this dissertation and allows the transparent synchronisation of the distributed simulations and all the agents within each of them.

Additionally this `internal_datetime` serves as bridge for including external agents that federate time across distributed simulations. **Another of the important contributions of this Execution Environment layer, is that synchronisation between distributed simulations is achieved by default at the moment in which external environments are mask as agents.** As mentioned in Section 3.6 distributed

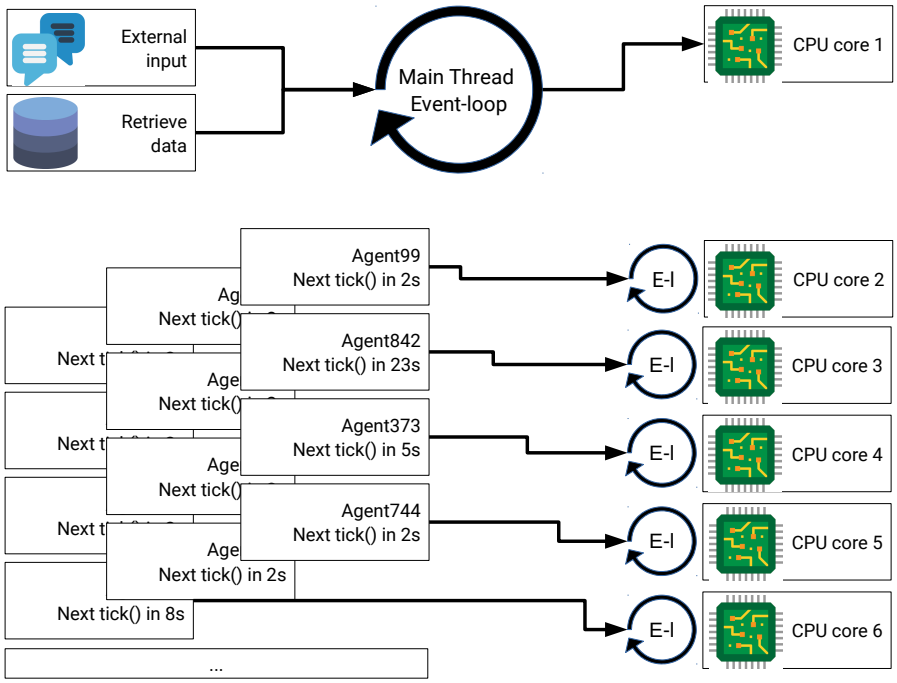


Figure 3.15: Mixed strategy approach for scheduling agent ticks. (Source: own research)

environments running in external computers are modelled here as empty behaviour agents which are updated from the outside. The empty agents have a `internal_datetime` that does not advance (because they have no behaviour) until the update takes place from the external environment through the *Sockets module* Section 4.5. The Execution Environment will treat it as another agent and make all agents wait until the external agent reaches this environment's time. Within this approach it is not needed to deploy additional synchronisation mechanisms (as Anylogic or Jade do) nor asking what time does the external environment have (as FIPA does). The GeoWorldSim environment tackles distributed time federation with this simple and straightforward mechanism.

In turn the masked agents could introduce new problems if the distributed systems are waiting for messages from others to advance their `internal_datetime`. More precisely, in distributed simulations, where messages determine the turn each simulation needs to advance, there might be cases of interlocking if not well coded or a message is lost. A simulation may stay in a loop with no time advance (e.g. waiting for data, because of an erroneous calculation) and consequently freeze the rest of the systems.

Figure 3.16 depicts three distributed simulations (`Simulation A`, `Simulation B` and `Simulation C`) connected to execute the same time. `Simulation A` has the slowest agents and `Agent-A1` is currently ballasting the advance of simulations `B` and `C` time-window. In this case, `Simulation B` has a wide time-window which allows other more advanced agents to receive execution time even though the simulation is waiting for `Agent-A1` to update its `internal_datetime`. By cons, `Simulation C` has a narrower time-window so while waiting for `Agent-A1` to advance its agents have already surpassed the highest time instant that will be *ticked*.

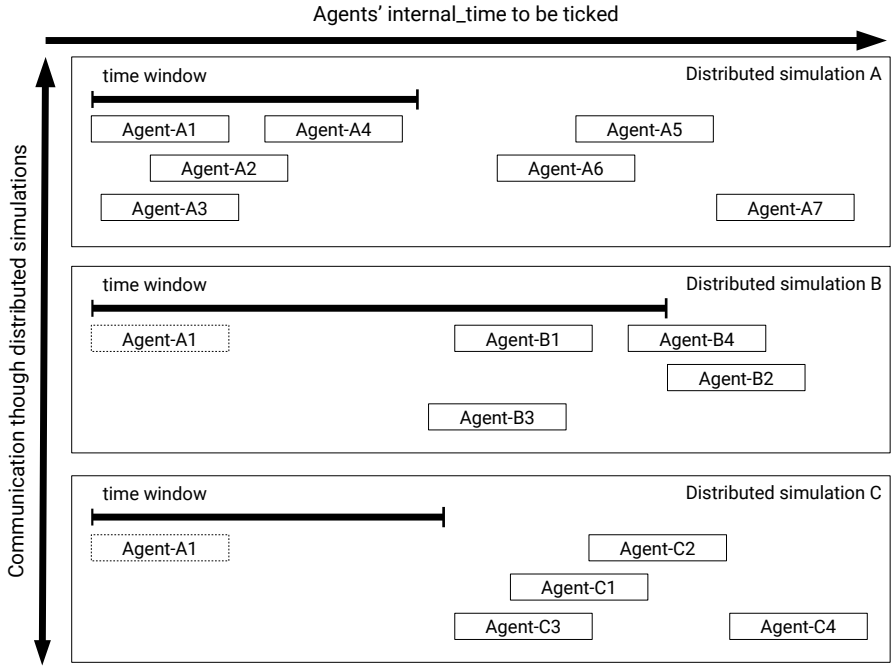


Figure 3.16: Distributed simulation synchronisation by using masked environments. (Source: own research)

```
Data: CURRENT_DATETIME from Time Environment
Data: MIN_TICK = CURRENT_DATETIME
Data: WIDTH of the time-window
for eventloop in eventloops do
    for agent in eventloop.agents do
        if agent.BUSY == 0 then
            CURRENT_DATETIME = MIN( CURRENT_DATETIME ,
              agent.INTERNAL_DATETIME )
        end
    end
end
if MIN_TICK < CURRENT_DATETIME then
    DELAY-TIME-ENVIRONMENT( MIN_TICK )
end
for eventloop in eventloops do
    for agent in eventloop.agents do
        if agent.INTERNAL_DATETIME < MIN_TICK + WIDTH and agent.BUSY
          == 0 then
            scheduled = CALCULATE-SCHEDULED( MIN_TICK , WIDTH ,
              agent.INTERNAL_DATETIME )
            eventloop.TRIGGER( scheduled , agent.tick() )
        end
    end
end
call_again = CALCULATE-ALL-FINISHED()
this.TRIGGER( call_again , this.TICK() )
```

Procedure tick function of the Execution Environment. (Source: own research)

CHAPTER

4

GeoWorldSim ecosystem

Once fully described the simulation core, it is time to connect the simulations to the world. Although the initial attempts of the platform were conceived as a single software containing all the functionality built-in, going to a full scalable product required an effort in modularisation. We humans are limited when managing complex problems all at once and the same happens when trying to design a code that does everything. It is well-known in software development that decomposing a complex problem into a number of simpler individual problems helps building piece-by-piece the desired solution and improves specialisation, maintenance and clash avoidance.

Throughout this time, the software has gone through different stages until becoming a full web-based, distributed and general purpose Geosimulations framework. The GeoWorldSim simulation core, has been surrounded by nine enabler modules (which some have already been mentioned briefly) that take secondary functionality out from the simulation engine to third party proven technologies. Each module is dedicated to independently covering one basic need in evaluating urban developments. Table 4.1 identifies all the modules of the proposed GeoWorldSim platform

together with Figure 4.1 that depicts the overall architecture of the framework and how all the parts communicate.

Within this chapter all the secondary modules that compound the GeoWorldSim platform are described along with how each supports a necessity for running complex Geosimulations. The following sections describe these nine parts that are key to round up the needs of the framework and provide:

Table 4.1: GeoWorldSim modules. (Source: own research)

Module	Task
Simulations Launcher	Web management of simulations
Authentication	Identity manager
Statics	File storage
Datasources	Data harmonisation
Sockets	Real time data bus for communication
Historical	Time history storage
Alerts	Notification
Intelligence building	Graphically building complex operations
Dashboards	Visualisation of executions and results
Simulations	Geosimulations core

Simulation launching: Section 4.1 presents the web Simulation Launcher for online managing executions.

User Authentication: Section 4.2 introduces the module designed for identity and profiles management serving as user database for the rest of the modules.

Statics: Section 4.3 presents a file storage service.

Datasources: Section 4.4 describes the module responsible for the harmonisation of different data sources (e.g. databases, files, Application Programming Interface (API)) into a single format to avoid creating many importers in the simulation core.

Sockets: Section 4.5 introduces the a communication bus for message sharing across distributed simulations and others.

Historical: Section 4.6 presents a novel mechanism for storing the results of simulations allowing time searches and richer knowledge extraction.

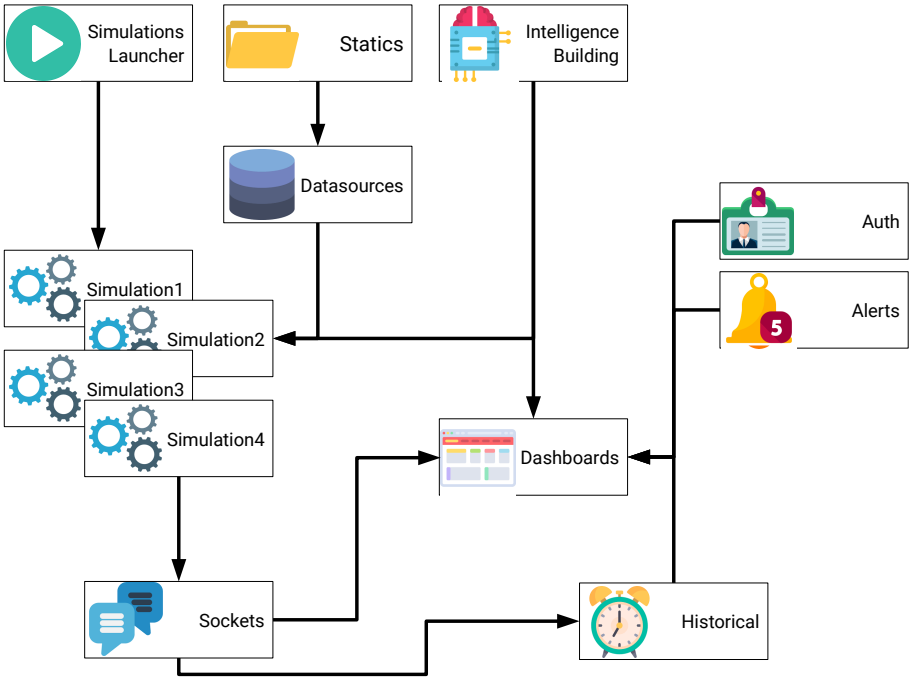


Figure 4.1: GeoWorldSim platform architecture. (Source: own research)

Alerts: Section 4.7 introduces a simple user alert storage for all modules to report errors and messages into a single endpoint.

Intelligence building: Section 4.8 presents a powerful tool for programming complex operations without needing to code by chaining functionality blocks.

Dashboards: Section 4.9 describes the graphical user interface module to create custom visualisations of running simulations and results.

4.1 Simulations Launcher

As mentioned in Chapter 3, while originally it was though to develop GeoWorldSim as a single desktop software, soon the necessity to be able to run several simulations

through the web and in parallel managing the collision in data emerged. Besides, like any web based software, there was the need for a server that constantly waited ready for receiving new requests and giving response to them. Taking into consideration simulations need to be created and destroyed, soon became clear the necessity to create an upper level process that orchestrated these Geosimulations.

The Simulations Launcher is an active enabler to bring to life, manage and kill simulations acting as a process-dispatcher. The Simulations Launcher deploys a web server with an interface and a REST API to answer user needs in terms of simulations. Through the web interface, a user can send the configuration to start a new simulation, view or destroy existing ones.

It has been coded in C++ for which the community has developed several tools to take this language to the web environment. This is the case of a third party library called *QHttp* (Amir, 2018) for opening a Hypertext Transfer Protocol (HTTP) server and receiving requests. These serve as protocol for invoking all operations a traditional web server would provide such as returning Hypertext Markup Language (HTML) rendered pages, files or executing commands. By matching requests to their processes, this module provides a REST API for simulations orchestration described in Table 4.2.

Using C++ allows easy operative system process creation and management. The Simulations Launcher contains a folder with all the available simulation executables and launches them according to the received requests. To parameterise a simulation, the API must receive a configuration JSON payload with at least the following parameters:

`target`: Simulation type that wants to be run to fetch the executable file.
`name`: Given to a simulation for better identification.
`start`: UNIX time-stamp of when the simulation takes place.
`speed`: Number from 0 to 1000 representing the speed of the Geosimulation.

This configuration can be extended for each type of simulation since the received JSON payload will be passed to the simulation executable for it to parse additional values.

When requested to start a new Geosimulation, the launcher firstly assigns the simulation's unique identifier that is globally used across all Geosimulation modules. It is also in charge of registering the identifier in all additional modules, that is, creating a database in Historical module, opening the communication bus in Sockets module and creating two dashboard to see the simulation functioning and results in Dashboards module. Once done, Simulation Launcher will prepare a folder with received configuration files and redirect the simulation's working directory, standard output and console to this folder. Thereby, all simulations write their results and console messages in an isolated and restricted folder, in spite of sharing the same executable file. Finally, the launcher will open a terminal and run the simulation executable in a detached way.

Table 4.2: Simulation launcher API. (Source: own research)

HTTP Request	Functionality
`GET /`	Renders the index page
`GET /view/:type/:view`	Returns any of the additional sub-views rendered
`GET /api/login/`	Log in using the Authentication module
🔒`GET /api/logout`	Log out
🔒`GET /api/user`	Get the logged in user's profile and specific attributes in this module.
🔒`GET /api/simulations/offset/:o/limit/:l`	Get short information of all my simulations
🔒`GET /api/simulation/:id`	Get all the information about a simulation
🔒`GET /api/simulation/:id/file/:file`	Get a file from the simulation working directory
🔒`POST /api/simulation`	Create a new simulation.
🔒`DELETE /api/simulation/:id`	Kill the execution of a simulation but maintaining its result files.
🔒`DELETE /api/simulation/:id/all`	Kill and remove all data about a simulation.

4.2 Authentication module

The authentication module centralises all aspects involving user profiles and their access to Geosimulations, including secure and private authentication and Single Sign-On (SSO). The Authentication module is the only component that stores all users of the platform, assigns their identifiers (ID), relates which modules they have access, and provides foreign services to access personal data providing a secure environment.

This module has been developed on top of Django (Holovaty and Kaplan-Moss, 2009), an open source development framework which allows creating web servers written in Python. It comes with built-in modules for programming the business logic, designing the datamodel, database interaction and user authorisation, automatically deploying the processes and user interfaces to manage them. Regarding the databases, Django supports communication with main engines thus providing an abstraction of the model and allowing to migrate from one system to another with relative ease. This makes Django the most appropriate option to build modules where complex business logic and database models are needed.

Django's built-in user authentication system, which already handles user accounts, groups, permissions and cookie-based user sessions, has been extended. This reduces the effort for account creation and management, as it supports the enforcement of policies and procedures for user registration, user profile management and the modification of user accounts. This built-in user authentication system allows administrators to quickly create customised pages for users, registration of tenant applications with access to user profile data and the handling of error notifications. Additionally, the default user data model has been extended by adding Suites, Applications and linking them for users to access to (as shown in Figure 4.2). Suites are aggregates of applications that serve to cover several needs of a common goal. Applications are all the modules described here and, according to the Suite they are associated with, users will have access to or limit on the use of said modules.

For users to SSO across all modules, the sharing of the authorisation, identification and role privileges is done through OAuth2 protocol (Yang et al., 2016). This is integrated within the previously described Django user system. Whenever an

anonymous user tries to log in another module of the Geosimulation ecosystem, the module derives the authentication process to this central login service. Doing so, full user information (e.g. name, address, phone, e-mail) is only stored in here while, remaining modules will only store a unique identifier for this user together with the locally needed attributes.

The Authentication module is also in charge of providing the user navigation bar described in Appendix B.0.1, globally used in all graphical user interfaces. This is so, because each navigation bar is personal since it contains the links to all the accessible suites and applications for a user. This module therefore, enables an API endpoint for obtaining the rendered navigation bar as an injectable piece of code. This HTML code comes with controls for logging in, out and links to all suites and services assigned to the user providing a unified look and feel over all the Geosimulation modules.

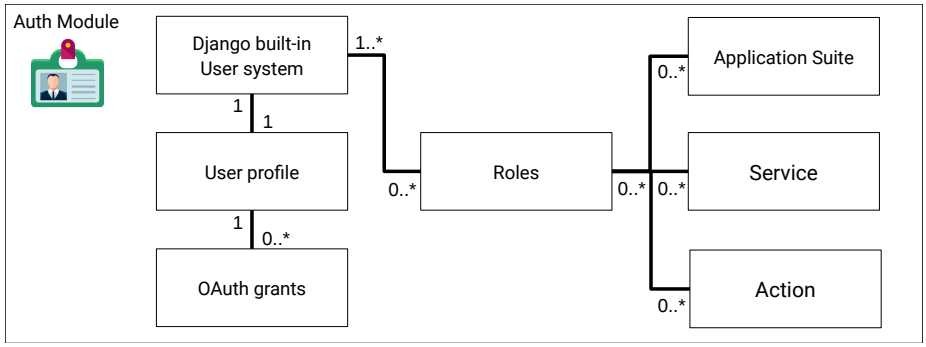


Figure 4.2: Authentication module model. (Source: own research)

Table 4.3: Authentication module API. (Source: own research)

HTTP Request	Functionality
`GET /`	Renders the index page
`GET /view/navbar`	Renders the navigation bar for the logged in user
`GET /view/navbar?uuid=:id`	Renders the navigation bar for the requested user
`GET /o/authorize/`	Oauth standard call, must send `client_id`, `response_type` and `redirect_uri` parameters
`GET /o/token/`	Oauth standard token refresh endpoint
🔒`GET /api/user`	Get the logged in user's profile.
🔒`GET /admin`	Enter Django's built-in user management system.

4.3 Statics module

This component is a global file storage built to centralise where to put files resources from all the modules and websites of the Geosimulation ecosystem. It is intended to host and retrieve static files providing a user security layer when uploading and making the data globally available for downloading. The Statics server is private and simple module but one of the most used tools by the rest of components since all web files (e.g. Cascading Style Sheets (CSS), JavaScript, XML) and data files (e.g. Comma Separated Value (CSV), Shapefile format (SHP), JSON) are uploaded and read from here. It offers a simple API for file uploading, accessing and deleting files, described in Table 4.4.

Table 4.4: Statics module API. (Source: own research)

HTTP Request	Functionality
`GET /`	Renders the index page
`GET /view/:type/:view`	Returns any of the additional sub-views rendered
`GET /api/login/`	Log in using the Authentication module
🔒`GET /api/logout`	Log out
🔒`GET /api/user`	Get the logged in user's profile and specific attributes in this module.
🔒`GET /api/files/offset/:o/limit/:l`	Get all my files
🔒`POST /api/file`	Create a new file. Mandatory fields is the file itself and an optional name.
🔒`PUT /api/file/:id`	Update a file. Any of the above fields can be updated.
🔒`DELETE /api/file/:id`	Delete one file.

4.4 Datasources module

Data-set registration and harmonisation module, whose purpose is being able to globally access any piece of information converted to a homogeneous format, no matter how or where it is read from. Doing so, when a simulation requires data from different sources and formats (e.g. buildings from an online API, transport network from a database, travel matrices from a CSV file) it does not need to implement the corresponding readers and converters. This task is delegated to Datasources module and all the connectors developed to read from main used sources.

The module has been programmed using the NodeJS framework (Tilkov and Vinoski, 2010). Based on Google's V8 Engine, NodeJs is an open source application runtime environment that allows creating server-side applications in JavaScript. NodeJS takes Javascript coding to the server-side for easily programming scalable, lightweight and efficient backends. One of its most notable features and advantages, is the non-blocking I/O model which, on the one hand, requires all intensive operations to be performed asynchronously and, on the other, allows delegating operations to external tools and invoking methods once they have finished. This makes NodeJS a really powerful option to build tools that mostly need to dispatch and redirect requests among other services.

The Datasources module exposes both a GUI for registering new data-sets and a REST API (Richardson and Ruby, 2008) for other modules to retrieve said data in standard JSON format. Using the GUI users can go through the registration process by selecting a datasource type, configuring its parameters and uploading a data file if needed. The internal model is depicted in Figure 4.3. Any data-set registered in the Datasources module will be globally available for reading using a single endpoint: `/api/datasource/:id/read/offset/:offset/limit/:limit`. The endpoint allows data pagination with the `offset` and `limit` parameters and returns any dataset in the format depicted in Section 4.4.

The following description, details currently implemented datasource types, their needed parameters and how they internally work.

PostgreSQL: Required parameters are `HOST`, `PORT`, `USER`, `PASSWORD`, `DATABASE`, `TABLE`. Using the access point (`HOST`, `PORT`); authentication (`USER`, `PASSWORD`) and where to read from (`DATABASE`, `TABLE`), whenever the datasource is read the connector will firstly perform a count of all the existing records in the table and then a `SELECT * FROM <table> LIMIT <limit> OFFSET <offset>` SQL query.

PostGIS: Required parameters are `HOST`, `PORT`, `USER`, `PASSWORD`, `DATABASE`, `TABLE`, `GEOMETRY_COLUMN`. Follows the same process to connect and query as the PostgreSQL connector but this will add a geojson conversion to the geometry column `SELECT * , ST_ASGEOJSON( geom ) FROM <table> LIMIT <limit> OFFSET <offset>`.

OpenStreetMap: Required parameters are `BOUNDS`, `TAGS`, `NODE`, `WAY`, `RELATION`. Heads to OpenStreetMap (OSM) data download API (Overpass API) (Overpass API, 2018) and performs an Overpass QL query to retrieve geographical data. OSM data is structured in nodes, ways, and relations and metadata in key-value pairs. This datasource requires the bounds of the area, key-value pairs and type of the elements to look up for. Nodes will be transformed into geojson points whilst ways and relations will be transformed to lines or polygons depending if they are closed or not.

REST API: Required parameters are `URL`, `METHOD`, `HEADERS`. Using the access point (URL), when reading it will make an HTTP request (`METHOD`) and converts the result to JSON format. It is important to notice that this connector does not count and return the amount of existing elements. If pagination would like to be used with this connector it would be necessary to update the `URL` or `HEADERS` parameters to manually write the paginated request.

FIWARE (Moltchanov and Rocha, 2014): Required parameters are `URL`, `ENTITIES_TYPE`, `SERVICE`, `SERVICE_PATH`, `HEADERS`. It will query a FIWARE Context Broker access point (URL) with the service and path headers and the FIWARE query format for type querying `URL + /entities?type=type&offset=offset&limit=limi` If user authentication wants to be added, it can be done by adding user tokens to the headers of the request.

CSV file: Required parameters are `FILE`, `DELIMITER`. In the registration phase the user will provide the CSV file and the module will upload it to the Statics server. Once stored, whenever the datasource is read, it will retrieve the file from Statics module and parse it using the specified delimiter character.

JSON file: Required parameter is `FILE`. Just the same as with CSV files, the origin file will be uploaded to Statics module and when reading, in this case the file will be downloaded and returned verbatim.

```
{
    "count" : 100,
    "data" : [
    { "id" : 4430 , "name" : "Cafe Plaza" , "type" : "cafe" ,
      "geom" : {"type":"Point","coordinates":[-5.921,36.1908]} },
    { "id" : 1968 , "type" : "parking" ,
      "geom" : {"type":"Point","coordinates":[-5.9256,36.1950]} },
    { "id" : 4261 , "name" : "Tony's" , "type" : "bar" ,
      "geom" : {"type":"Point","coordinates":[-6.1469,36.4186]} }
    ]
}
```

Code 4.5: Example response of executing : /api/datasource/DS23/read/offset/0/limit/3. Notice that although the count attribute telling that there exist 100 records, the limit parameter has requested only 3 of them. (Source: own research)

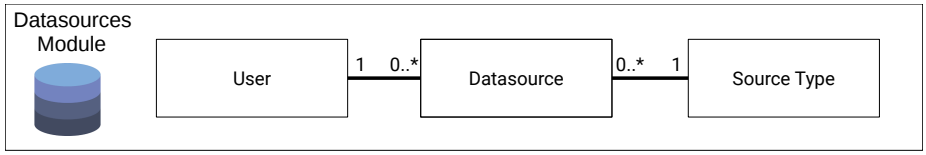


Figure 4.3: Datasources module model. (Source: own research)

The module has been designed ready for adding new datasource types extending its capabilities. It contains a `/static/sourcetypes` directory where new types can be added by creating a new folder and putting an `icon.svg` and `read.js` files in it. The icon file serves as a representative symbol for faster datasource type

identification and the `read.js` file contains the Javascript asynchronous function to be executed when reading the source type. This process strongly relies on Javascript's Async/Await, Promise and Eval techniques. For a better understanding of these techniques, please consult Appendix A. Every datasource type has the same function headers and implements its inner process to gather and transform the data. Thereby, the asynchronous function and Promise all source types must comply with in their `read.js` file is depicted in Listing 4.6.

```
module.exports = async function( params={}, offset=0, limit=100 ) {
    return new Promise( ( resolve , reject ) => {
        try {

            // Implement the reading procedure

            // If success return the result using the resolve callback
            let results = { data : [] , count : 0 };
            resolve( results );

        } catch ( err ){

            // If error return the message using the reject callback
            reject( err );
        }
    });
}
```

Code 4.6: Datasource reading function (`read.js`) all types must comply with. (Source: own research)

To briefly summarise, the functionality exposed by the module is collected in Table 4.5.

Table 4.5: Datasources module API. (Source: own research)

HTTP Request	Functionality
`GET /`	Renders the index page
`GET /view/:type/:view`	Returns any of the additional sub-views rendered
`GET /api/login/`	Log in using the Authentication module
🔒`GET /api/logout`	Log out
🔒`GET /api/user`	Get the logged in user's profile and specific attributes in this module.
🔒`GET /api/datasources/offset/:o/limit/:l`	Get all my datasources
🔒`POST /api/datasource`	Create a new datasource. Mandatory fields is the datasource name, type and parameters.
🔒`PUT /api/datasource/:id`	Update a datasource. Any of the above fields can be updated.
🔒`DELETE /api/datasource/:id`	Delete one datasource.

4.5 Sockets module

GeoWorldSim's simulation core provides the main source of information for the rest of the modules. Additionally, as said, the decision to avoid spending any effort in rendering inside the simulation engine was adopted for it to be fast and only dedicated to emulating the behaviour of agents. Thus, Geosimulations need a way to spread the state of their agents into other modules. To this end, the Sockets module enables a publisher-subscriber (Eugster et al., 2003) pattern based on Websockets for each simulation to open its own communication channel.

The WebSocket Protocol (Fette and Melnikov, 2011) is a widely supported open standard included in the latests HTML specifications for creating full-duplex connections. It is widely used in real-time web applications where data wants to be pushed from the server without needing a client request. Previous methods for simulating full-duplex connections were based on polling (Loreto et al., 2011), a synchronous method wherein the client makes a request to the server to see if there is any information available. Polling is acceptable for cases with fixed time interval of message availability. However, it requires the client to open and close many unnecessary connections or holding requests open until there is some information to be sent (e.g hanging-GET or pending-POST techniques).

Websockets take traditional socket usage to the web environment, allowing persistent and low latency connections that support transactions originated by either the client or the server. When connecting, both sides make a handshake over TCP Protocol in which the clients informs it wishes to upgrade to the WebSocket Protocol. This issue is not always documented in web socket libraries and requires extra tuning the web environment and proxies to support the `upgrade` header. Once the connection has been established, messages can be sent back and forth using the methods defined by the WebSocket simple interface.

The WebSocket standard is simple. A WebSocket object has only `send()` method; `close()` method; and triggers events on opening, closing, message arrival and error occurring. Over these, a service protocol has been designed to spread agent changes and notifications, all modules must fulfil. The Geosimulation Sockets pro-

tocol establishes that all message must contain a `signal`, `socket_id` and `body` fields:

`signal`: It comes to represent the reason of the message. Currently allowed values are:

- **`join`:** To request joining, that is subscribing to, the communication bus of a simulation.
- **`entity`:** To communicate a newly updated entity inside the `body` payload.
- **`message`:** To inform using plain text about the state or error of a simulation inside the `body` payload.

`socket_id`: Unique identifier (ID) of the socket, matches the simulation's unique identifier who emitted this message. It is useful when receiving messages from many sources.

`body`: JSON object with the main content of the message. When receiving a `join` signal, it is expected to receive a `socket_id` parameter containing the simulation to listen to. For the `entity` signal, a fully JSON-LD serialised entity is expected following the agent ontology described in Section 3.2.1. Finally, with `message` signal a simple string is expected as body.

When opening a socket, a client can send a `join` signal with the identifiers of the simulations it wants to receive messages from. Geosimulation Sockets module enables connecting to as many busses as desired and receiving communications from different sources through the same socket. Figure 4.4 depicts the message flow between the communication bus and subscribers. Notice that the Sockets server does not perform any user credentials validation, so a valid simulation ID is enough to join and watch data sent through a simulation's socket. Malicious users may try to broadcast fake values to the modules involved. This is why this module is hidden from the user and interaction is done through others. Even so, simulations IDs use 128 bits long numbers based on the algorithm of Leach et al. (2013) whose probability of being guessed is rather low. This is a minimum risk in favour of simplifying and speeding up information sharing over the bus.

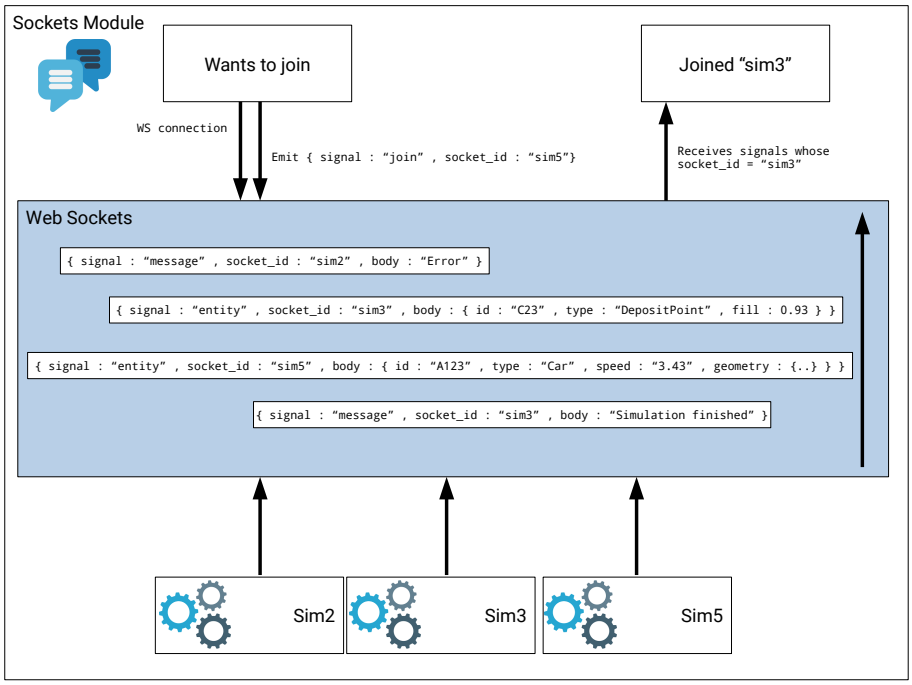


Figure 4.4: Sockets data flow. (Source: own research)

Table 4.6: Sockets module API. (Source: own research)

HTTP Request	Functionality
`WS /`	Connect a websocket using the WS protocol.
`GET /api/socket/:id`	Get if a socket exists or not. Existing sockets will return 200 HTTP code and none existing ones 400 HTTP code.
`POST /api/socket/:id/:signal`	Send a message through that socket emitting the said `signal`. Requires a JSON object as payload to be sent as socket message body.

4.6 Historical module

The already mentioned Sockets module provides a way of knowing the current state of a simulation at the moment it takes place. However, users require to go back and forth in time or watch a geosmulation days after this has concluded. Answer to this requirement is given by the GeoWorldSing Historical module.

This tool connects a NodeJS server to an Influx DB (Persen and Winslow, 2016), a time-series platform designed to handle metrics and events. Time series are simply measurements or events that are tracked, monitored, downsampled, and aggregated over a period of time. The key difference with time series data from regular data is that queries will always ask questions about an element over time. With the arrival of the IoT paradigm, these databases are uniquely positioned to solve the challenges of millions of events coming in, filtering and analysing. A time series consist of a set of values where time is a meaningful component of the data. Using a timestamp as a primary first class attribute, Influx DB allows easily knowing and searching through the state of an element over the time. Geosimulations time series data amount is enormous and very concentrated in a short period of time. This scale has been one of the primary drivers behind the creation of specialised stores which need to face the challenges in:

Write throughput: As said, urban simulations can send thousands of new values per second during their execution. Inserting such amount of data into a regular relational database like MySQL and PostgreSQL, quickly knocks over most systems without careful tuning. Even then, relational databases have limits for the amount of data a system can accept at one time.

Query throughput: One aspect of time series data is that new data is almost always more valuable than the old data. The real-time nature of time series makes it necessary to expose last received data in queries as fast as possible. Additionally, most time series data can be summarised into intermediate values since trends provide more information than individual data points. Time series engines must provide mechanisms and optimisation for these operations.

Influx DB is a high performance Time Series Database. It is designed for storing hundreds of thousands of points per second and has its own InfluxDB SQL-like query language built specifically for time series. The Historical module makes use of the Influx DB engine to store the entire course of a urban simulation. It specialises in processing timestamped data and implements optimisation to address data managing. Although Influx DB sets up its own REST API, an intermediate NodeJS dispatcher was developed for defining an own API and adding WebSocket communications. Updated and Inserts in the module are done both using the API described in Table 4.7 or whenever an `entity` signal is received through its simulation socket. Thereby, all agent evolution is recorded into its own independent database and made available for back and forth searching any state in time.

Table 4.7: Historical module API. (Source: own research)

HTTP Request	Functionality
`POST /api/simulation/:id`	Create database for historic storage of a simulation.
`DELETE /api/simulation/:id`	Delete the simulation database with all its agents.
`GET /api/simulation/:id/entities`	Get a list of what agent types exist in the historic simulation data.
`GET /api/simulation/:id/entities/:type`	Get the last status of all the agents of type entity_type.
`/offset/:o/limit/:l`	
`GET /api/simulation/:id/entities/:type/start`	Get the timestamp of the first agent stored.
`GET /api/simulation/:id/entities/:type/end`	Get the timestamp of the last agent stored.
`GET /api/simulation/:id/entities/:type/at/:a`	Get all the agents (that existed) at a certain timestamp. The :at parameter by default is assumed to be in nanoseconds. Include a duration literal at the end of the epoch timestamp to indicate a precision other than nanoseconds. For example in JS: new Date().getTime() + 'ms'
`POST /api/simulation/:id/entity/:type/:id`	Notify a new state of an agent. Agent data must be sent inside the request payload. For every agent notification, an additional attribute called time will be created storing when the entity changed to this status. However, if this time attribute wants to be overwritten, a *time* attribute can be explicitly added in the payload.

4.7 Alerts module

Simple tool for storing emerged alerts and messages in their own database. It provides a mere endpoint where users can list received messages, discard and forward them to an email or API. Developed using NodeJS this module deploys a very simplistic API and datamodel where every alert/message is composed of:

Status: Flag for storing whether the alert has been attended or it is still pending.
Level: Number for describing criticallity of the message.
Title: Title of the message/alert.
Description: Description of the message/alert.
Generator type: Type of who generated this message/alert. It can be a simulation or the Logic building module.
Generator id: Id of who generated the message/alert, for the user to be able to identify the source with the generator type and id.
View link: Optional link referencing the element who generated the message/alert.

Table 4.8: Alerts module API. (Source: own research)

HTTP Request	Functionality
`GET /`	Renders the index page
`GET /view/:type/:view`	Returns any of the additional sub-views rendered
`GET /api/login/`	Log in using the Authentication module
🔒`GET /api/logout`	Log out
🔒`GET /api/user`	Get the logged in user's profile and specific attributes in this module.
🔒`GET /api/alerts/offset/:o/limit/:l`	Get short information of all my alerts
🔒`GET /api/alerts/status/:s/offset/:o/limit/:l`	Get short information of all my alerts that have a given status
🔒`GET /api/alert/:id`	Get all the information about an alert
`POST /api/alert`	Create a new alert. Mandatory fields is the alert level, title and description. Open for other modules to generate alerts.
🔒`PUT /api/alert/:id`	Update an alert. Any of the above fields can be updated.
🔒`DELETE /api/alert/:id`	Delete one alert.

4.8 Intelligence building module

A complex Logic building tool designed to build data processes in a visual and intuitive way. This module is able to receive input event data in real-time, make a series of calculations and generate instantaneous data/alerts in reply to changing conditions. It comes in relation to reactive programming (Czaplicki and Chong, 2013), a paradigm concerned with data flows and the propagation of change. While standard computation tend to employ synchronous, request-response interactions, reactive programming aims at creating observers which are subscribed to data changes. Whenever a change is emitted, the observer will perform calculations to process the change and obtain more meaningful information about a situation. This calculus can include analysing the sequence of the value changed, aggregating all the values or comparing with some thresholds (e.g. for generating alerts).

The GeoWorldSim Logic Building module is a complex tool for creating asynchronous functions, called *Behaviours*, that chain simple operations blocks. Each code block has several input and output parameters which need to be connected to their prior and post. When a *Behaviour* function is invoked, it will go through all its blocks executing them in cascade and passing the parameters from one to the next. These functions can be triggered in response to events, such as changes to data in the Historical module, or invoked by an HTTP call from an agent in a simulation. Furthermore, if needed, there are function blocks for making requests out of the tool to complete or notify some of their calculations. On the whole, the Logic Building module provides a really sophisticated mechanism for business intelligence implementing and asynchronous computing delegation.

The data-model shapes the construction of these *Behaviours* through *Blocks* and *Links*. fig. 4.5 depicts the data-model and its classes:

User: Gathered the main information from the Authentication module, here additional configuration parameters are stored.

Behaviour: Users can create as many behaviours as required. A behaviour is a concatenation of blocks that will be sequentially executed passing parameters from one to another.

Block: Collection of pre-coded procedures that serve as a basis for behaviours to combine them. A block contains a set of configurable meta-parameters and a piece of code that performs its duty.

Category: Blocks are grouped into categories according to their purpose for users to better find them.

BehaviourBlock: Describes the code-block to be used within a behaviour and in which position of the sequence.

BehaviourLink: Links an output of a block to an input of another block. The blocks linked do not need to be continual; that is, an output of a very early executed block in the sequence, can be sent to a block several positions later in the sequence.

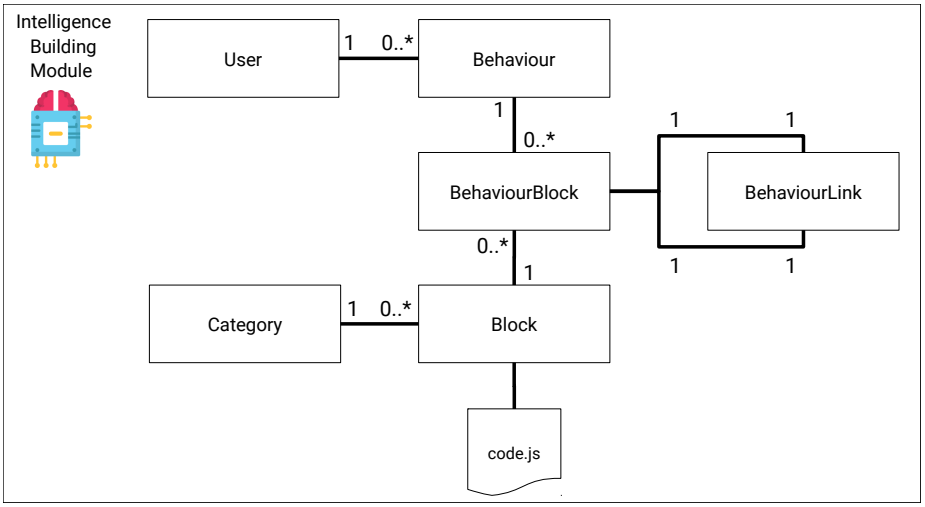


Figure 4.5: Intelligence building module data-model. (Source: own research)

Built using NodeJS, the tool enables users to visually drag and drop blocks of code and construct what a function should compute and return. fig. 4.6

Invoking a *Behaviour* will result in executing all its *Blocks* in cascade. Firstly it orders all execution indexes (as shown in section 4.8) and then enters a loop that:

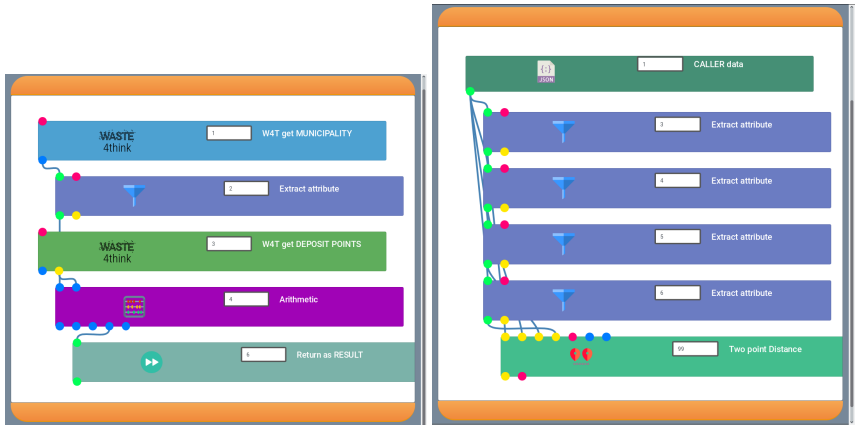


Figure 4.6: Examples of two operation that request outer information, process them and return the result to an agent. (Source: own research)

- Gets the next behaviour block.
- Gets the link to go and fetch the values from previous executed blocks (as depicted in Section 4.8).
- Sets an object with all the input values.
- Asynchronously loads and executes the *Block* code (as described in Section 4.8 and appendix A).

```
let indexes = [];
let outputs = {};

global.BehaviourBlock.findAll({ where : { behaviour_id : behaviour.id } ,
                                order : [[ 'index' , 'ASC' ]] })
.then( blocks => {

    for(var i in blocks){
        indexes.push( blocks[i].index );
    }

    return indexes.reduce( (promise, index) => {
        return promise
            .then( () => {
                return behaviour.executeBlock( req , index  , outputs );
            })
            .then( result => {
                outputs[ index ] = result;
            })
            .catch( console.error );
        }, Promise.resolve( outputs ));
})
.then( () => {
    res.send( { data : outputs[ indexes[ indexes.length - 1 ] ] } );
})
.catch( err => {
    res.send( { error : err } );
});
```

Code 4.7: Behaviour execution code that asynchronous and sequentially executes each block. (Source: own research)

```
Behaviour.prototype.executeBlock = function( req , index , previous_outputs = {} ){

    return new Promise( (resolve, reject) => {

        const behaviour = this;
        let behaviour_block;
        let links;

        global.BehaviourBlock.findOne({
                where : { behaviour_id : behaviour.id , index : index } ,
                include : [{ model : global.Block , as: 'block' }] })
        .then(bb => {

            behaviour_block = bb;
            return global.BehaviourLink.findAll({
                where : { behaviour_id : behaviour.id , to_index : index } });
        })
        .then(ls => {

            links = ls;

            let input = behaviour_block.values || {};
            links.map( l => {
                if( outputs[ l.from_index ] ){
                    input[ l.to_input ] = outputs[ l.from_index ][ l.from_output ];
                }
            });

            return behaviour_block.block.execute( req , input );
        })
        .then( output => {
            resolve( output );
        })
        .catch(e => {
            reject(e);
        });
    });
}
```

Code 4.8: Each Block execution that fetches linked values from previous outputs. (Source: own research)

```
Block.prototype.execute = function( req = {} , input = {} ){
const block = this;

return new Promise( (resolve, reject) => {
    try {

        require( path.join(
                __dirname, '..',
                'static' , 'blocks' ,
                block.path , 'code.js' )
        )
        ( resolve , reject , req , input );

    } catch( err ){
        reject( { error : err } );
    }
});
}
```

Code 4.9: Block asyncrhonous and dynamically code loading and execution. (Source: own research)

4.9 Dashboards module

All data visualisation has been decoupled into its own analytics and monitoring tool. The Dashboards module is the main road for users to interact with data of almost every part in the Geosimulation ecosystem providing a configurable viewport for the information produced, collected or transformed. It is though for users to configure multiple views, known as dashboards bringing different points of view of the system state.

According to the dashboards nature, they are built to connect and display the information of with different modules of the platform:

Live mode: Dashboards for displaying the current state of a simulation. They make intensive use of the Sockets module for gathering changes in the attributes of agents and display them in different meaningful visualisations. The most common way to represent a simulation's state is by drawing a map with the agents interacting in it but this dashboards can also display charts and aggregation of values (e.g. visualising in real time the current position of vehicles, the filling level of deposit points or the power demand in substations).

Historic mode: Intended to viewing all the evolution of a simulation enabling going back and forth in time. They allow setting a start and end date between which loading all the data or slide through time updating agents to the values they had back then. The most common representations will consist of maps displaying the agents interaction but also time charts and graphs are available (e.g. going back and forth in simulated citizens water demand, comparing the initial and finish state of a simulation, watching the complete route performed by a vehicle, etc.).

Event mode: Designed to monitor a certain event or attribute value which is significant for the user or simulation carried. It uses the Sockets module for listening the changes in the value to be monitored (e.g. listing substations whose power demand is reaching a security threshold, locating in a map deposit points that have been moved from their original position).

Alert mode: Dashboards for including information about pending alerts and provide a gateway for users not needing to directly head to the Alerts module.

To accomplish easy construction and configuration of these, the Dashboards module is based on NodeJs application with a simple but powerful data model depicted in fig. 4.7. Each of the classes has the following assignment:

User: Gathered the main information from the Authentication module, here additional configuration parameters are stored.

Dashboard: Users can create as many dashboards as required. A dashboard is a simple HTML container to provide a layout for its widgets.

Widget Template: Collection of pre-designed templates that serve as a basis for widgets to perform functions such as display key performance indicators, filter dashboard results or visualise data using interactive charts or tables. A template along with the configuration of its parameters, is what gives rise to a widget. To this end, every widget contains a folder with three indispensable files:

`config.html`: When a user chooses the template of what type of widget it wants to create, the `config.html` view will display a predefined form to configure the required fields of the template. This `config.html` is as much representative as possible of the later appearance the widget will have.

`prepare.js`: Thought as a middle data processing point before the final rendering of the widget. Some widgets may require big amounts of data and complex processing in order to create aggregations or graphs. To this end, before calling the `render()` function that transfers the configuration parameters to the final rendered widget, this `prepare.js` can perform some additional calculations also sent to the `render.html` file.

`render.html`: The final HTML view of the *compiled* widget to be rendered by the web browser. Depending on the widget it can just present data received from the configuration and `prepare.js` or it could perform its own requests to modules and third party API to gather information.

Adding a new template and folder automatically exposes them for users to create new widget types.

Widget: The basic building blocks of a dashboard. Choosing one of the templates, a user can create as many widgets as wanted. A widget is just composed by its parameterisation and the reference to its template. Widgets can be reused across different dashboards, that is the reason why they do not directly belong to a single dashboard.

Widget position: Relates a widget with a dashboard. As widgets can be used in different dashboards, this class relates a widget with a dashboard and adds some additional parameters such as `width` and `height` of said widget in the dashboard. The widgets position will be auto adapted to the dashboard's viewport using the new CSS Grid layout mechanism (Atkins Jr et al., 2013) and different `width` and `height` according to device size (as shown in Figure 4.8).

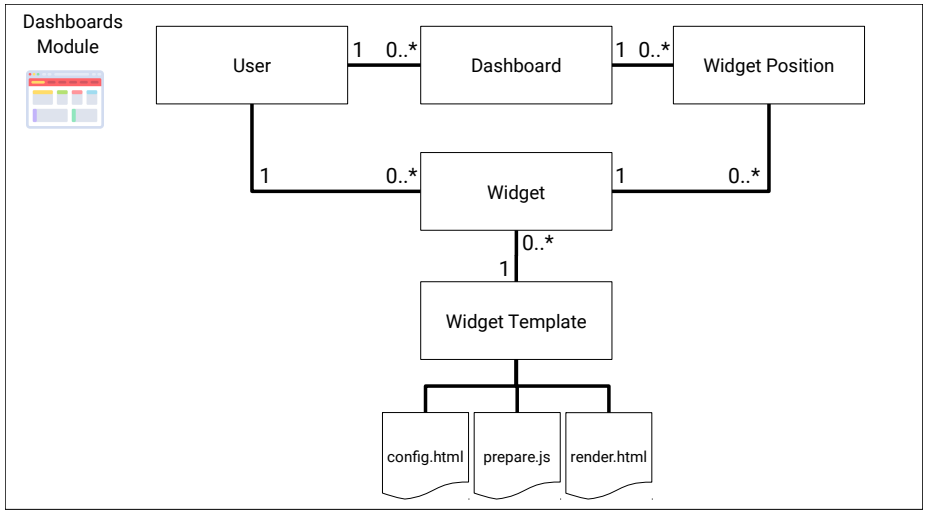


Figure 4.7: Dashboards module data model. (Source: own research)

In the dashboard designer, users can add and create the widgets by drag-and-dropping the templates to the desired position. All the dashboard customisation

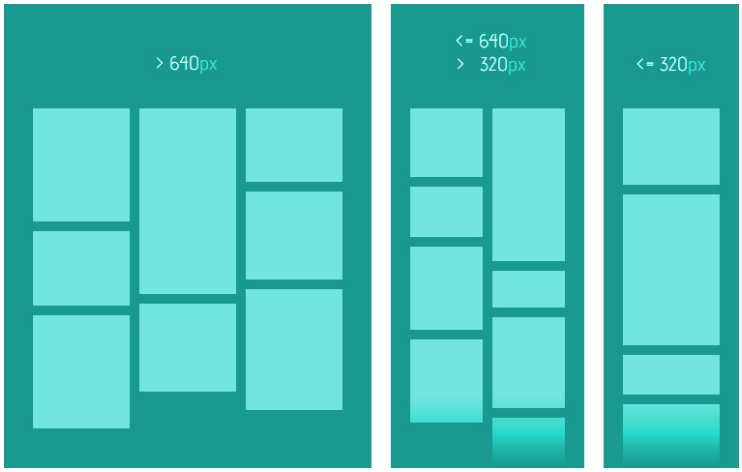


Figure 4.8: Responsive grid layout. (Source: own research)

and widget parameterisation process has been programmed using the methodology detailed in Appendix B. fig. 4.9 shows the visual aspect for creating, managing and final result of an example dashboards.

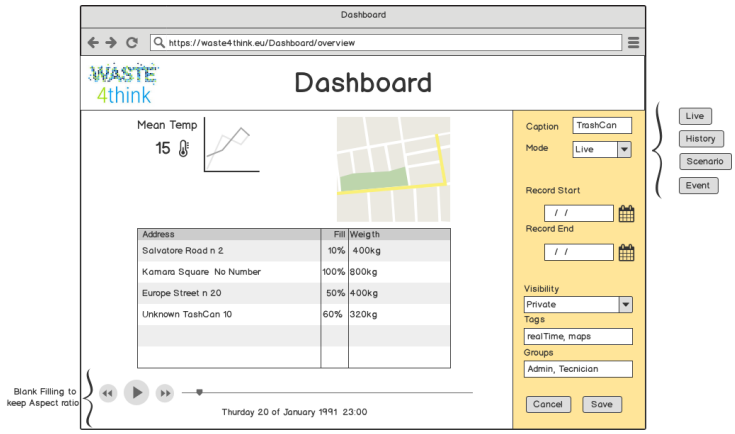


Figure 4.9: Example of a dashboard. (Source: own research)

CHAPTER

5

Use-case 1 - Ubiquitous Smart Cities

Intelligent environments are thought to be a key resource in assisting older adults and people with disabilities through their daily life by following the ambient assisted living paradigm. The use of sensors and computing devices allows intelligent environments to monitor the activity of inhabitants and plan suitable interventions depending on the activities and behaviour of the monitored person.

There are many challenges to face in the area of intelligent environments and human activity recognition, such as the type and layout of the sensors to be used, or the definition of accurate algorithms for activity recognition and behaviour modelling (Kafalı et al., 2014), (Cvetković et al., 2016). However, many of the research directed to address these issues finds a common operational problem which, in the best case, slows down the research, hindering the capability of running meaningful experiments in intelligent environments.

This operational problem has already been identified by many researchers (Helal et al., 2012), (Rashidi and Cook, 2011), (Azkune et al., 2015), (Buchmayr et al., 2011). Intelligent environments are expensive to build and maintain. Whenever there is a need to test new sensor layouts, it is generally difficult to reconfigure the environment, thus researchers have to be very careful with the initial design. In addition, the existence of a proper intelligent environment does not guarantee the generation of needed data-sets, since recruiting humans for experiments is a troublesome process. In consequence, many of the data-sets generated in intelligent environments do not allow to test and verify complex theories concerning human behaviour (Helal et al., 2012).

The solution to this problem is not trivial. In this line, many researchers have proposed the development of several simulation approaches for the definition of intelligent environments. Simulated intelligent environments offer many positive features, e.g. the generation of large data-sets, total control of the environment and sensor layouts, cost-effective experiments and ability to define very specific experiments. Synnott et al. (2015) divide simulators into two main groups: model-based simulators and interactive simulators. The purpose of this chapter is to present a model-based simulator to mitigate the problems of experimenting in intelligent environments.

Specifically, **this chapter presents a Geosimulation built using the GeoWorld-Sim for human activities in sensorised spaces.** The advantage of this approach is the feasibility to parameterise different scenarios and evaluate how the agents behaviour and interactions are affected each time. This Geosimulation's scope goes far beyond the replication of human activities and is extended to test how different smart city sensoring layouts affect the smart city infrastructure. This methodology allows a myriad of researchers to benefit from it, for cases such as the evaluation of electrical distribution systems, adequate sizing of water and sewerage networks, accessibility studies or traffic simulations. Section 5.3 introduces a rigorous methodology to validate the activity data-sets generated by the simulator which also aims to be used by other researchers to assess the validity of their approaches.

The objective of our simulator is to provide a useful tool to generate all the required realistic data for specific research questions. In this chapter, a Geosimulation is tested for activity recognition data generation in two scenarios:

- Single-user scenarios, where only one person is being monitored. Sequential and overlapping activities can be performed. The majority of the research in intelligent environments has traditionally been focused on such scenarios, thus the importance of proving the validity of the tool in these conditions.
- Multi-user scenarios, where multiple persons are monitored in the same intelligent environment. Those scenarios are more challenging. However, the full potential of GeoWorldSim is shown in multi-user scenarios, due to the agent-based approach adopted. To the best of our knowledge, this is the first simulator to address the multi-user scenario for activity datasets.

5.1 Modelling social behaviour

Following the categorisation presented by Synnott et al. (2015), simulators for intelligent environments can be classified into two main groups:

- **Model-based approaches:** these approaches use activity models, which can be obtained by diverse ways, to create synthetic data-sets that store information about the activation of sensors during the execution of activities.
- **Interactive approaches:** a human user can interact with virtual environments and sensors set up by the simulation; depending on the actions executed by the user, the state of the virtual environment and sensors changes accordingly, storing that information in a synthetic data-set.

For a complete review of both approaches, it is recommended to read the work of Synnott et al. (2015). Since this paper relies on the definition of a specific model-based approach, the related work will be focused on exploring this perspective.

Model-based approaches for synthetic data generation using simulators demand the specification of activity models explaining the generation of events over time, the

probability of events occurring, the time taken for each event during the performance of specific activities and the definition of the behaviour of monitored virtual people as a generator of activities over time.

An early example of a simulator called DiaSim was developed by Bruneau et al. (2009). The DiaSim simulator executes pervasive computing applications by creating an emulation layer and developing simulation logic using a programming framework, where the models are defined. It is more focused on simulating applications such as fire situations, intrusions and so on to identify potential conflicts, hence it cannot be used for human activity recognition. In contrast, using GeoWorldSim makes it possible to model human activity/behaviour and simulate these type of situations for analysis.

Helal et al. (2011) developed a simulator called Persim, which has been enhanced in the new version Persim-3D (Helal et al., 2012). Persim is an event driven simulator of human activities in intelligent environments. Persim is capable of capturing elements of space, sensors, behaviours (activities), and their inter-relationships. The simulator allowed users to define activities by specifying the sensors involved in each activity, the order of sensor activations, the maximum and minimum typical sensor values and activity duration. Based on these parameters a list of sensor data could be generated in the Sensory data-set Description Language. In contrast with the Geosimulation presented hereby, Persim only offers single-user single-activity scenarios, activities are defined as fixed sequences of actions and dependencies between activities cannot be modelled.

An earlier example of a model-based simulator was provided by Bouchard et al. (2010). They developed a simulator called SIMACT, where the activity models are captured using a form-based interface for users. Those forms allow to specify scripts that detail the series of steps involved in the performance of activities within an environment. SIMACT users could define behaviour related parameters such as the order of events, the time taken for each event and the objects involved in the event. Besides, users could also define actions associated with the completion of each step which modify the state of the environment. Activity scripts could be replayed in real-time or fast-forwarded and could be replayed with adjustments to event timings.

While SIMACT only applies an activity-based approach, the proposed Geosimulation follows the idea of Persim of a sensor-based approach for simulation, which is more realistic and provides richer information.

A different approach was presented by Mendez-Vazquez et al. (2009), who demonstrated the use of Markov chains describing the order of events, combined with Poisson distribution to calculate a range of realistic activity times and probability distributions to calculate a range of sensor values to generate a simulated activity data-set. Their tool considered activities such as reading, sleeping, walking and sitting together, adding interesting metrics including time and energy expenditure. A similar approach was shown by Okeyo et al. (2012), who use a synthetic data generator tool to simulate time intervals between sensor activations. Their research is focused on sensor data stream segmentation, so the tool generates varying patterns of sensor activations in order to verify their approach.

Kormanyos and Pataki (2013) developed a simulator to model the activity of a single inhabitant within an intelligent environment. Individual behaviour profiles allow modelling concepts like typical sleep amount, and the change in current state such as thirst and tiredness, explicitly introducing the state of the monitored person as an actor in the activity generation process. Their simulator was capable of outputting data from simulated motion sensors, RFIDs and water consumption.

Finally, Azkune et al. (2015) presented another model-based simulator, where varying activity executions, time-lapses and sensor error models are included. They provide an original way to capture realistic activity models using surveys to target users.

The major problem of model-based simulators is their dependence on activity models. This implies that the quality and accuracy of the resulting data-sets, heavily relies on the quality of the activity description model and the associated parameters. However, obtaining accurate activity models is complicated, since human behaviour is very complex and full of details which are hard to capture. The work presented hereby, builds on the experience of previous research in the field of human behaviour by simplifying the process of the creation of activity data-sets. The purpose is to provide a sensor-based simulation platform that generates activity data-sets based

on previously defined expert definitions of human behaviour. By Geosimulating the interaction between occupants and sensors when carrying out daily activities, accurate data-sets can be produced to then be used not only for human activity recognition -the main target of this work-, but also in diverse research domains, such as simulating the energy demand within a building, proper sizing of water and energy networks or load balancing.

5.1.1 Human Simulation Model

The ability to simulate human behaviour within intelligent environments, such as smart homes or heavily monitored habitats, is closely related to the definition of a descriptive model that outlines the interactions between human actions and the entities that are receivers of those actions, all the while preserving the nature of decision making, inherent to human needs and desires. In this context, the human simulation model conceived is defined by the concepts of *Person*, *Intelligent Environment*, *Behaviour*, *Activity*, *Action*, *Object* and *Sensor*. Figure 5.1 shows an overview of these concepts along with the relationships among them.

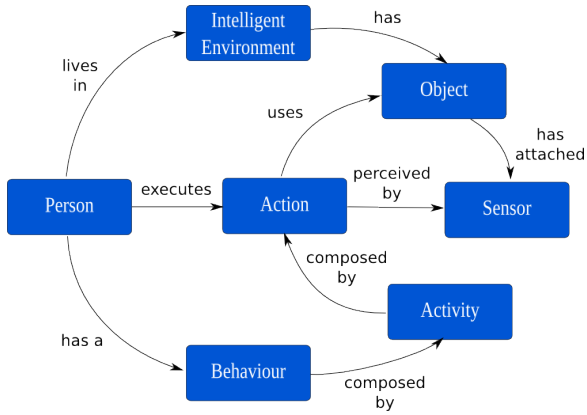


Figure 5.1: Conceptual diagram for the Human Simulation Model. (Source : own research)

The *Person* is the central axis of the behaviour model, an entity that lives in an *Intelligent Environment* (smart home, smart office, or even a smart city) whose definition can be as complex as desired, and is governed by spatial and temporal laws and constraints, such as the existence of physical barriers, sensors' detection ranges, availability of open/closed roads, business calendars, saving light periods, etc. Specifically, for this research, our *Intelligent Environment* represents a smart home that houses a set of inanimate objects with specific *Sensors* attached to them. The designed methodology currently follows the dense sensing monitoring approach which relies on miniaturised simple sensors installed on *Objects* of interest, such as cabinets, drawers, or taps, that trace a *Person*'s interaction through a Boolean state (i.e. activated/deactivated).

Objects, on their part, may also have their own set of constraints. For example, when a *Person* opens a door (executes action `open` over an object of type `door`), the state of the door changes to `open` and thus, can not be opened again by another person until the state of the door changes again to `closed`. The main idea behind this is the desire to maintain simplicity in the objects' behaviour, really focusing on the interaction initiated by a person or another object.

Each *Person* has a specific *Behaviour* that defines the characteristics of its interaction with the *Intelligent Environment*. As already mentioned in section 3.4, the behaviour model follows the well established hierarchy of *Behaviour*, *Activity* and *Action* (Chen et al., 2012) which helps define the way in which the person performs the *Activities*, in regards to order, relationships, and priorities according to both the person's personal preferences and the constraints of the *Intelligent Environment* in which the person lives.

5.1.2 **Parameterising the Human Behaviour Model**

The selection of which activities to carry out during the day and the order in which to perform them is driven not only by the person's needs and preferences, but also by any event that might take place in the intelligent environment and directly influence the person's behaviour. From the moment a person wakes up, they have a list of daily

activities that must be fulfilled by the time the day comes to an end. The person will then try to complete as many activities as possible, performing, at least, the most essential in the impossibility of performing all of them. Therefore, the key point in emulating a person's behaviour is the capability to complete as many daily activities as possible, taking into consideration different aspects and fitting them to the person's needs.

To this end, this Geosimulation extends GeoWorldSim's behaviour model with new attributes and capabilities based on three main concepts:

- **Activities:** These model every single possible activity in the use case. Activities consist of a set of basic attributes that define their priorities, their approximate duration or the conditions for their completion. Some examples of activities include *wake up*, *prepare dinner*, *do laundry* and *take shower*. In addition, each activity has a set of associated *actions* that establish the sensors that can be activated during the activity execution. For example, performing the *preparing dinner* activity may involve the execution of several actions such as *open cupboard*, *open fridge*, and *use microwave* which are then perceived by the corresponding sensors, as shown in Table 5.1.

Table 5.1: Example of activities and their corresponding actions. (Source : own research)

Activity	Actions
Preparing breakfast	Open drawer | Open fridge | Open cabinet | Use microwave
Washing dishes	Open drawer | Open cabinet | Use dishwasher
Shower	Use toilet | Use shower faucet HOT | Use shower faucet COLD
Preparing dinner	Use oven | Use drawer | Use cabinet | Use microwave | Use fridge
Do laundry	Use door | Use hamper | Use washing machine | Use laundry dryer

- **TODO list:** Every person has a list of all the activities they need to perform during the day. This list can be extended automatically since there are activities that will add prerequisites and others that will generate new needs, e.g. the *preparing dinner* activity generates a new *washing dishes* activity, even though

it was not originally defined in the person's TODO list. The TODO list is reset every single day.

- **DONE list:** Every person keeps track of the activity that is being done at each moment, as well as the activities that are already finished. It is necessary to point out that the execution of certain activities may remove others from this list. For example, after cooking food, a person will add the *cooking activity* to the *DONE* list. However, when the person executes the *eat* activity, the *cooking* activity will be removed from the *DONE* list, since the execution of another *cooking* activity will be necessary if the agent wants to eat again.

The definition of these concepts is heavily influenced by the Belief-Desire-Intention paradigm (BDI) which provides mechanisms for separating the process of selecting a plan from the execution of other currently active plans (Velleman and Bratman, 1991). In general, people tend to perform similar activities throughout the day, though they do not execute them in the same order, invest the same amount of time, or give them equal importance. When selecting an activity over others, there are several factors that come into play, such as the priority of the activity, the amount of time it is expected to take, any possible time restrictions (e.g. the person must perform a *go to work activity* at a certain hour, so all the morning activities, or at least the most important ones, should be finished by then), or even the existence of mandatory preconditions must be fulfilled before the activity is carried out (e.g. the *take shower* activity must be performed before the *dressing* activity). In order to allow greater flexibility, I have selected those attributes that, to our consideration, influence most the selection process. The attributes that parameterise each type of activity are shown in Table 5.2.

As stated in Section 5.1.1, each activity defines its own set of actions that are directly related to the interaction between the person and the objects living in the intelligent environment. When an activity is chosen for execution, the person performs the actions within the activity. The execution of several actions is what defines the degree of completion of an activity. Note that for an activity to be completed, performing each and every single one of its actions is not always necessary. For example, *preparing lunch* is related to actions such as *use oven*, *use microwave* or

Table 5.2: Definition of activity attributes. (Source : own research)

Name	Description
Type	Alphanumerical code that identifies the activity type.
Attendance	Attribute for describing whether a person has to be physically present during the execution of the activity. An attended activity will monopolise the attention of the agent until it finishes (i.e. *prepare dinner*), whereas an unattended activity will allow the agent to just initiate it and carry on with other activities (i.e. *watch TV*). This attribute is key in enabling a concurrent behaviour among agents and a more realistic representation of an intelligent environment where the simultaneous activation of several sensors may take place.
Estimated duration	Amount of time the activity will take, according to what the person estimates, even though there may be some deviations in reality
Priority	Number that determines the importance an agent gives to a certain activity. This priority is modified according to the preferences of the person and the compliance with the daily schedule.
Desirable start time	Indicative start time at which the person would like to start performing the activity. If possible, the agent will try to schedule the start of the activity at this time.
Desirable end time	Indicative end time at which the agent would like to have the activity already finished. If possible, the agent will try to schedule the end of the activity before this time.
Mandatory start time	Compulsory time at which the activity must start. Necessary to model time critical activities such as going to work or watching a certain TV show.
Mandatory end time	Compulsory time at which the activity must finish.
Preconditions	Activity codes for modelling prerequisites, i.e. other related activities that must be already finished by the time this activity is chosen. These codes will be checked against the person's *DONE* list to ensure their fulfilment. If there are pending preconditions, these missing activities will be added to the *TODO* list and the person will choose once again which activity they will perform.
Outcomes	Performing some activities may lead to the appearance of other new activities, like needing to *take a shower* after *doing exercise*, or *toileting* after *having lunch*. For this purpose, there is a list of activity codes so that when an activity finishes, these new outcomes are appended to the person's *TODO* list.

use burner, but they are not always used every time the activity is executed. In order to differentiate and characterise how they help fulfil their associated activities, the Geosimulation developed specifies a set of particular attributes for each action, as shown in Table 5.3.

Table 5.3: Definition of action attributes. (Source : own research)

Name	Description
Duration mean	Average duration of the action
Duration deviation	Deviation of the duration of the action.
Finish increment	Amount of completion fulfilled with the execution of this action.
Frequency	Defines how much an specific action within an activity is prone to be executed.
Associated sensor	Sensor attached to the household appliances and that gets activated every time a person uses them.

5.1.3 **Human activity data-sets**

In order to validate the methodology in both single-user and multiple-user paradigms, I considered several reference data-sets. After a careful analysis, the data-sets that were finally selected were those collected by (Tapia et al., 2004) and (Cook and Schmitter-Edgecombe, 2009), since they present two different implementations of dense sensing monitoring approaches and the distribution of the sensors throughout the household is accurately specified.

- **Single-user:** The data-set chosen to evaluate the single-user case provides information about the activation of sensors located inside two different single-person apartments, with about 77 and 84 reed switch sensors installed for monitoring around 16 activities of daily living (see (Tapia et al., 2004) for detailed information). The first subject (DS_1) was a 30-year-old professional woman, whereas the second subject (DS_2) was an 80-year-old woman. Both subjects spent most of their time at home and lived alone in their one-bedroom

apartments. Once the sensors were installed, they started collecting data for 14 days. During the study, the subjects used a custom annotation tool to build a detailed record of their activities. The data-sets are formatted in comma separated files (CSV), where for each activity, the day, start time and end time as annotated by the subject appear. Additionally, for each annotated activity the activation and deactivation times for the involved sensors are listed.

- **Multiple-user:** The data-set chosen to evaluate the multiple-user case provides information about the activation of sensors located in the WSU smart apartment testbed (Cook and Schmitter-Edgecombe, 2009) during the 2009-2010 academic year. Specifically, the data-set provides information from 24th August 2009 to 1st May 2010. At that time, the apartment housed two residents, R1 and R2. Sensor events are annotated with the corresponding activity that was being performed, distinguishing among 13 activities for R1 and 12 for R2. WSU apartments rely on a slightly different monitoring approach, stressing the usage of motion sensors all along the apartments. However, they also use object-attached sensors.

Since both single-user and multiple-user scenarios are addressed, I believe the combination of these data-sets provides a solid reference to evaluate the performance of the Geosimulation in terms of human behaviour modelling. Furthermore, it is also shown how it can model different intelligent environments, ranging from the heavily sensorised apartments from (Tapia et al., 2004) to the lighter sensorised and motion-based monitoring approach from (Cook and Schmitter-Edgecombe, 2009).

In a pre-processing stage, the data-sets where analysed to identify the most relevant information that helped building a human behaviour model. The objective is that, when the behaviour model is executed in the Geosimulation, the resulting data-set is similar to the original in terms of frequency, duration and hourly activation of the sensors. First of all, the sequence of activities, time-slots and mean duration for each data-set were monitored, so as to identify any behavioural patterns in terms of the order in which the activities were executed and the sensors that were activated each time. Table 5.5 contains an example of the activity model extracted from the reference data-sets, used as input.

Table 5.4: Format of the RAW data-sets used for validation. (Source: (Tapia et al., 2004))

timestamp	id sensor	sensor	action	activity
2003/03/27 06:43:40	67	Cabinet_67	ON	Toileting
2003-03-27 06:43:43	67	Cabinet_67	OFF	Toileting
2003-03-27 06:44:06	100	Toilet Flush_100	ON	Toileting
2003-03-27 06:44:20	101	Light switch_101	ON	Toileting
2003-03-27 06:44:35	57	Medicine cabinet_57	ON	Toileting
2003-03-27 06:44:36	58	Medicine cabinet_58	ON	Toileting
⋮	⋮	⋮	⋮	⋮
2003-04-11 22:24:17	71	Drawer_71	ON	Grooming
2003-04-11 22:24:17	71	Drawer_71	OFF	Grooming

Table 5.5: Pre-process data-sets used for validation. (Source : own research)

activity	sensor	freq	mean (s)	deviation (s)
$activity_1$	$sensor_1$	f_1^1	μ_1^1	σ_1^1
$activity_1$	$sensor_2$	f_1^2	μ_1^2	σ_1^2
⋮	⋮	⋮	⋮	⋮
$activity_1$	$sensor_n$	f_1^n	μ_1^n	σ_1^n
⋮	⋮	⋮	⋮	⋮
$activity_m$	$sensor_1$	f_m^1	μ_m^1	σ_m^1
⋮	⋮	⋮	⋮	⋮
$activity_m$	$sensor_n$	f_m^n	μ_m^n	σ_m^n

5.1.4 Validation Methodology

From now on, the data-set generated by the Geosimulation will be named as *simulated data-set* and *reference data-set* to the data-set that it is wanted to replicate. The validation methodology is based on the following definition:

Definition 1 (Similarity)**.** *Two persons are considered to behave similarly if and only if:*

- *The frequency and duration of every action within a certain activity are analogous.*
- *The temporal distribution of the actions throughout the day is analogous.*

Within the sense of *similarity*, I differentiate between two concepts: assessment of the *internal consistency* and *coherence with real measurements*.

- **Assessment of the internal consistency.** The objective is to verify whether Geosimulations are able to build data-sets that emulate the behaviour of a certain activity model. For this analysis, I first built an empirical distribution of the frequency and duration of the activation for each sensor available in the reference data-sets. After defining a *TODO* list based on the activity patterns of each subject, I ran several simulations to generate simulation data-sets by recording the person's interaction with sensors through the execution of actions within activities. The frequency and duration values obtained from the simulations were then analysed to verify whether they lay inside the confidence intervals for the expected values given by the reference data-sets. Since the latter did not follow any particular distribution, the confidence intervals were built using bootstrapping methods (Davison and Hinkley, 1997). In addition, the existence of about 100 sensors made it compulsory to take into consideration the *look-elsewhere effect* (Randall Munroe, 2011), overcome with the use of the Bonferroni correction (Derrac et al., 2011).
- **Coherence with real measurements.** The objective is to verify whether the simulator can replicate the reference data-set by following the evaluation methodology proposed by Helal et al. (2011). In this case, I built a *TODO*

list that detailed the activities to be performed for a period of 14 days. In this context, the simulated and reference data-sets are *similar* if the distributions of frequencies, duration and distributions of sensors' activations are equivalent (Robinson et al., 2005) with respect to a *region of similarity*, i.e. the percentage of differences that can be considered *experimental errors*. This region should be defined following expert advice or other (not statistical) procedures (Walker and Nowacki, 2011). Since there is no clear procedure to fix this value within this context, the relation between the length of the region of similarity and the number of not similar sensors is plotted. Doing so provides more trustworthy results over other tests like those of Kolmogorov-Smirnov or Anderson-Darling, since these methods can only assess the dissimilarity, but never the similarity. After the region of similarity is defined, two one side confidence intervals for a statistic are tested whether they lie within the region of similarity. In this case, the null hypothesis is rejected and can be concluded that both samples come from the same population, allowing to conclude the existence of equivalence. Finally, as in the previous case, the Bonferroni correction is used to overcome the *look-elsewhere effect*.

5.2 Modelling physical reality

Regarding the physical layout in which the behaviour model has been tested, the Geosimulation recreates the dense sensing monitoring approach (Chen et al., 2012), which relies on miniaturised simple sensors installed on objects of interest, such as appliances, urban equipment, power generation plants, etc. These sensors are able to trace invocations and can monitor boolean states (i.e. activated / deactivated) as well as continuous values, such as those given by power consumption or power generation. It is worth mentioning that objects may also have a behaviour. For example, when a person turns on an appliance (executes action `run` over an object of type *appliance*), the state of the appliance changes to `running` and thus, it can not be started again by another person (until the state of the appliance changes again to `stopped`).

Following this approach several types of appliances have been defined:

Urban equipment: These appliances refer to the devices deployed outside the houses and along public spaces, such as lamp posts and recycling containers. Generally, this type of devices is associated with a public service. For example, a waste container will issue a notification when the fill level is above a certain threshold, or a lamp post will only be functioning during specific hours.

Common Appliances: These are tools that have a constant behaviour and need to be manually started and stopped. Most of these appliances are used to carry out activities that involve being in contact with the device and not leaving it unattended, such as using an oven, a shower, or a sink faucet. Even though there are people that may overlap the use of more than one common appliance at a time (e.g. listening to the radio while having a shower or prepare a coffee without turning off the television) it will be assumes that all people turn off an appliance before carrying out another activity.

Countdown Appliances: These appliances have the capacity of auto stopping themselves, such as a microwave or a toilet flush. Therefore the activities involving these appliances can be parallelised with other tasks since the initiator does not have to stop them manually.

Program Appliances: Some Countdown Appliances may have different behaviours throughout the cycles that make up their scheduling program. That is the case of washing or coffee machines. This sequence of actions is emulated by scheduling the activation and deactivation of inner components and having each of them associated to a specific power consumption.

Idle Appliances: Appliances such as fridges or boilers usually remain idle, monitoring some properties to verify whether they stay under a certain threshold. Once this condition is surpassed, the appliance starts working and tries to bring that property back to the desired state.

Finally, several appliances of Program and Idle type also have a behaviour. Appliances that emulate a certain intelligence are the ones that can be used by the demand-response controller. The behaviour has been added to those appliances which execution can be delayed to any other moment of the day, like for example times at which the energy grid is not saturated or the energy price is lower.

5.2.1 Generation/Consumption skills

In this way, the Geosimulation contains a set of skills upon creation that define its main core functionality. Taking a `Fridge` for example, this device is given a `Temperature` skill with methods to retrieve or change said temperature value. In addition, skills can act upon skills, such as `Heating skill` or a `Cooling skill` which can modify the `Temperature skill` by gradually simulating the increase or decrease in temperature over time.

Considering the demand-response scenario where power demand monitoring is the main pillar of this strategy, the set of skills assigned to an agent will not only modify the attributes of the agent, but also implicitly demand an energy load. In the `Fridge` example, the execution of a `Heating` or a `Cooling` skill will not only modify the temperature inside the `Fridge`, but will also generate a load, i.e. the energy needed in order to perform such change in temperature. The notation of the method is common to all consuming skills in which each device is supplied power and, depending on how the appliance works and what skills are defined, demands a specific amount of power.

5.2.2 Ideal architecture for a demand-response strategy scenario

Figure 5.2 shows the ideal architecture of a demand-response scenario and how each element is mapped to the simulation approach. The bottom layer of the architecture represents a city which energy demand is constantly monitored. Information regarding the energy consumption of devices within the city is sent to a common communication middleware which centralises all the information, registering all the events and notifying them to the Demand-Response Control. The Demand-Response Control analyses all the data received and provides a set of signals back to the monitored city so as to influence citizens behaviour with the objective of leveraging energy demand and energy production.

Following these schema, the developed Geosimulation emulates the behaviour of a city in terms of energy consumption, and evaluates the feasibility of deploying several communication architectures depending on the amount and type of devices

connected to the communication middleware chosen, in this case FIWARE. The development and implementation of the Demand-Response Control will be carried out in future research.

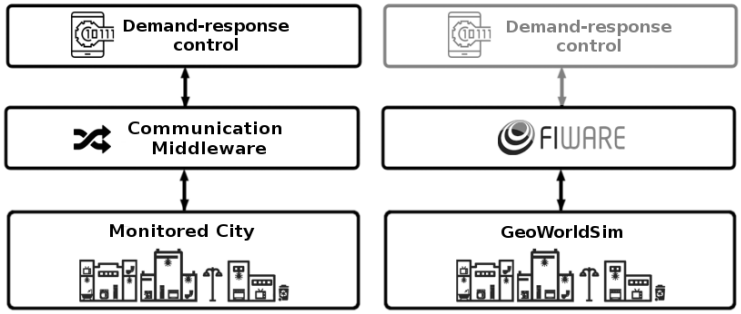


Figure 5.2: Real architecture and its simulated version. (Source: Own research)

FIWARE Middleware

Some GeoWorldSim simulations rely on FIWARE as the central middleware for creating a link between the monitored devices and the core of the demand-response control. FIWARE is a new cloud infrastructure created by the European Commission and several major European ICT companies for the development and global deployment of services and applications of the Future Internet (Consortium, 2016). It is based on a set of modules, known as General Enablers (GE), which provide services such as shared cloud computing resources, big data analysis, data management, modules for integrating web applications to the FIWARE infrastructure, and security control.

Figure 5.3 depicts a general overview of FIWARE's architecture, with the central core of the solution being the Orion Context Broker GE. This GE is a C++ implementation of the Publish/Subscribe mechanism, providing interfaces to the NGSI9 and NGSI10 API's with services such as registering context producer applications (e.g. sensor humidity within a room), update control information (e.g. send updates about changes in humidity), subscription and notification on changes on context information (e.g. when the humidity changes) or with a given frequency (e.g. re-

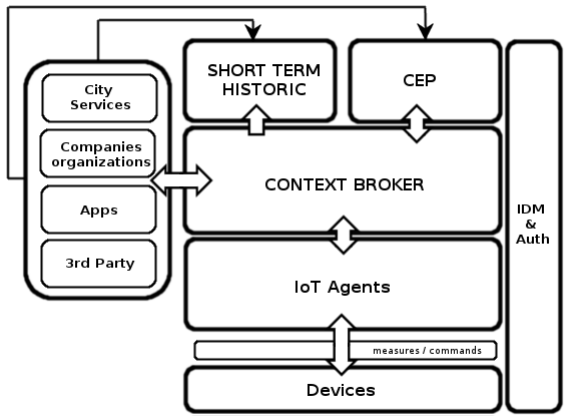


Figure 5.3: Diagram with the overall architecture of FIWARE (Consortium, 2016). (Source: FIWARE)

trieve humidity values every hour), and query context information about registered applications.

The Context Broker has the ability of managing Context Information at large scale by keeping virtual representations of physical devices. The interaction with the devices takes place on the event of modifying or updating their virtual representations in the Context Broker. When a petition of registration or update is performed, the APIs map each device to a specific Context Provider in order to retrieve all the necessary information through a Uniform Resource Identification protocol. Device types are grouped in domains according to their hierarchical schema and queries can be done in less or more deepness of the hierarchy. Section 5.2.2 describes in detail this hierarchical representation and the messages needed for communication.

According to FIWARE documentation, the proper way to connect any device to the platform is through the use of an entity called IoTAgent which bridges system specific communication with FIWARE middleware, regardless of the protocol and data representation. Specifically, in the system, a particular IoTAgent has been implemented in charge of binding the REST API synchronous calls of the Context

Broker to the asynchronous and non-blocking Qt Signal and Slots communication architecture.

Messaging format

Each FIWARE message must contain a minimum set of properties for the Context Broker to be able to identify the service, domain and device the message belongs to. Therefore, the smallest piece of information to be sent will on average be of 180 bytes.

Additionally, messages can describe other entity attributes, values and control commands. Each attribute must contain a name and type of value. An example of such control command can be seen in Source Codes 5.10–5.12.

```
1   {
2     "content-type": "application/json",
3     "X-Auth-Token": "[TOKEN]",
4     "Fiware-Service": "HouseManager",
5     "Fiware - ServicePath ": " / "
6   }
```

Code 5.10: Header example.

```
1  {
2    "devices": [{
3        "device_id": "[DEV_ID]",
4        "entity_name": "[ENTITY_ID]",
5        "entity_type": "thing",
6        "protocol": "IoTA-UL",
7        "timezone": "Europe/Madrid",
8    }]
9  }
```

Code 5.11: Payload example.

Entities are registered in FIWARE Context Broker using HTTP POST operations and sending the entire headers and payload description. Updates, however, are

```
{
  {
    "device_id":"[DEV_ID]",
    "entity_name": "[ENTITY_ID]",
    "entity_type": "thing",
    "timezone": "Europe/Paris",
    "endpoint": "http://{{DEV_IP}}:{{PORT}}",
    "attributes": [
        {
          "object_id": "t",
          "name": "temperature",
          "type": "int"
        }
      ],
    "commands": [
        {
          "name": "ping",
          "type": "command",
          "value": "[Dev_ID]@ping|%s"
        },
      ]
  }
}
```

Code 5.12: Control message example.

reported using a HTTP PATCH operation and just sending the headers together with the attributes that need to be updated, reducing the size of the bandwidth needed.

5.3 Human Behaviour modelling Results

For the *assessment of internal consistency*, a person's activity model was defined and ran 100 simulations to build the distribution of frequencies and duration of activation for all the sensors in each of the single-user and multiple-user scenarios. In both cases, every simulation configured a set of daily activities for each of the occupants living in the smart home. Even though any random activity model could have been chosen as reference, those obtained from the data-sets described in Section 5.1.3 are used.

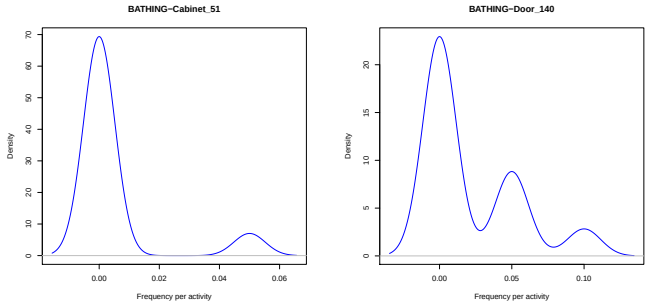
Single-user scenario

Figures 5.4 and 5.5 show, respectively, a distribution of the *frequency* and *duration* of activation of two sensors within the *Bathing* activity for the 100 simulations performed in the single-user scenario. Clearly, the plots are quite irregular and do not follow any standard distribution. Being a common issue among all the sensors identified in the reference data-sets for the single-user scenario, I needed to use bootstrapping techniques in order to build the confidence intervals needed to evaluate the accuracy of the data-sets generated by the Geosimulation. In particular, for DS_1 and DS_2, the attendance attribute gains importance, since several activities, like $WatchingTV$ or $ToiletFlush$, do not need for the person to be physically present, allowing the simulation of parallel activities.

Table 5.6 contains an extract of the results of one of the simulations carried out for DS_1. In this example, the duration of activation (column *duration*) from the reference data-set for all the sensors of the *Bathing* activity lie within the 0.05-confidence interval (columns *low C.I. (%)* and *upper C.I. (%)*). On the other hand, for 8 sensors, the frequency of activation (column *frequency*) from the reference data-set for the same sensors do not lay within the 0.05-confidence interval (columns *low C.I. (s)* and *upper C.I. (s)*).

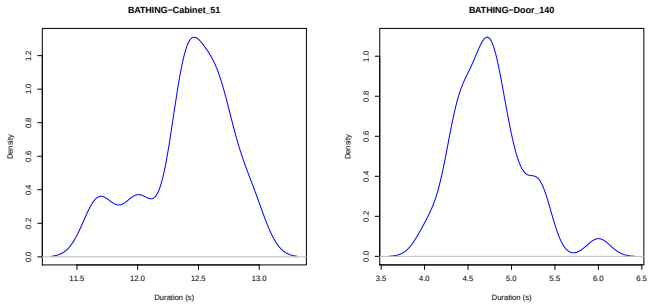
Overall, for DS_1, 94 % of the reference values for the frequencies of activation lie within the 0.05-confidence interval, whereas, 88 % do so for DS_2. Further, for

Figure 5.4: Distribution of the frequency of activation of two sensors in DS_1 (single-user scenario). (Source : own research)



(a) $Cabinet_{51}$ (b) $Door_{140}$

Figure 5.5: Distribution of the duration of activation of two sensors in DS_1 (single-user scenario). (Source : own research)



(a) $Cabinet_{51}$ (b) $Door_{140}$

DS_1, 98 % of the reference values for the durations of activation lie within the 0.05-confidence interval, while 99 % do so for DS_2.

Table 5.6: Internal Consistency – Example of the results for one of the executions on DS_1 (single-user scenario). (Source : own research)

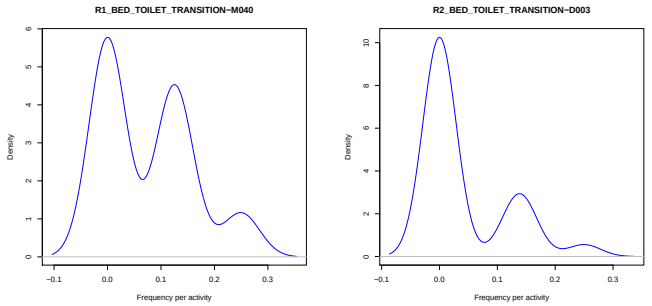
activity	sensor name	frequency (%)	low C.I. (%)	upper C.I. (%)	similar frequency	duration (s)	low C.I. (s)	upper C.I. (s)	similar duration
BATHING	Burner94	1.6	1.3	2.2	✓	2.2	2.1	2.3	✓
BATHING	Cabinet138	0.4	0.3	0.8	✓	20.5	19.2	21.1	✓
BATHING	Cabinet51	0.4	0.3	0.7	✓	13.1	12.3	13.7	✓
BATHING	Cabinet59	0.4	0.2	0.7	✓	41.6	39	44	✓
BATHING	Cabinet61	0.4	0.3	0.9	✓	12.7	12.5	13.8	✓
BATHING	Cabinet66	0.4	0.3	0.8	✓	41.9	39.1	43.3	✓
BATHING	Cabinet67	8.6	7.6	9.3	✓	79.7	79.1	81.8	✓
BATHING	Cabinet72	0.8	0.7	1.3	✓	18.1	17.4	18.7	✓
BATHING	Cabinet79	2.3	1.9	3.3	✓	3 611	3 559.7	3 728.6	✓
BATHING	Cabinet83	0.4	0.3	0.9	✓	23.2	22.3	24.9	✓
BATHING	Closet81	1.6	1.8	2.8	✗	10.6	10.4	10.9	✓
BATHING	Dishwasher70	0.8	0.6	1.3	✓	2 060	1 984.4	2 164.4	✓
BATHING	Door130	3.4	3.1	4.4	✓	141.4	138.4	145.5	✓
BATHING	Door140	1.6	1.4	2.3	✓	4.7	4.6	4.9	✓
BATHING	Door54	2.6	2.4	3.5	✓	2.5	2.5	2.6	✓
BATHING	Drawer62	1.6	1.7	2.6	✗	2 515.8	2 433.9	2 575.9	✓
BATHING	Drawer71	0.4	0.4	0.9	✓	7.4	6.9	7.7	✓
BATHING	Drawer75	0.4	0.3	0.9	✓	16.4	16	17.5	✓
BATHING	Drawer78	0.4	0.2	0.7	✓	2.1	2	2.2	✓
BATHING	Drawer82	2.4	2.3	3.3	✓	1 866.8	1 804.6	1 888	✓
BATHING	ExhaustFan96	4	4	5.2	✗	1 098	1 074.7	1 120.7	✓
BATHING	Freezer137	1.6	1.2	2.1	✓	726.9	700.5	740.7	✓
BATHING	Garbagedisposal98	0.4	0.2	0.6	✓	1	1	1	✓
BATHING	Jewelrybox139	1.1	0.8	1.6	✓	12.1	11.9	12.8	✓
BATHING	Lamp76	0.4	0.3	0.8	✓	7 164.2	6 732.4	7 374.7	✓
BATHING	Lightswitch101	4.4	3.9	5	✓	2 122.1	2 073.6	2 167.8	✓
BATHING	Lightswitch104	0.4	0.3	0.8	✓	2.1	2	2.3	✓
BATHING	Lightswitch108	0.4	0.3	0.8	✓	5 706.3	5 445.3	5 998.5	✓
BATHING	Lightswitch92	2	2	3.1	✗	3 675.1	3 589.2	3 807	✓
BATHING	Lightswitch95	0.4	0.4	0.9	✗	1.1	1	1.1	✓
BATHING	Medicinecabinet57	7.1	6.3	7.8	✓	2 538.6	2 482.1	2 581.8	✓
BATHING	Medicinecabinet58	5.3	4.8	6.5	✓	36.5	35.9	37.3	✓
BATHING	Refrigerator91	0.4	0.2	0.8	✓	18.2	17.3	19.1	✓
BATHING	Showerfaucet93	28.6	20.3	23.2	✗	1 083.6	1 059.9	1 090.9	✓
BATHING	Sinkfaucet-cold88	3.4	3.5	4.5	✗	13.4	13.1	13.6	✓
BATHING	Sinkfaucet-hot68	3.5	3.3	4.3	✓	12.8	12.5	13.1	✓
BATHING	Toaster131	0.4	0.3	0.8	✓	1	0.9	1	✓
BATHING	ToiletFlush100	4.6	3.4	4.4	✗	3 369.4	3 334.9	3 503.3	✓
BATHING	WashingMachine142	0.4	0.3	0.8	✓	34.3	32.6	36	✓

† Lower C.I.: Lower limit of the confidence interval
† Upper C.I.: Upper limit of the confidence interval

Multiple-user scenario

Figures 5.6 and 5.7 show, respectively, the *frequency* and *duration* of activation of two sensors for the 100 simulations performed within the *Bed toilet transition* activity for the two occupants of the smart home, $R1$ and $R2$. As in the single-user scenario, the plots do not follow any standard distribution. Being a common issue among all the sensors in the multi-user activity data-set, I used bootstrapping techniques in order to build the confidence intervals needed to evaluate the accuracy of the generated data-set.

Figure 5.6: Distribution for the frequency of activation of two sensors in DS_3 (multiple-user scenario). (Source : own research)

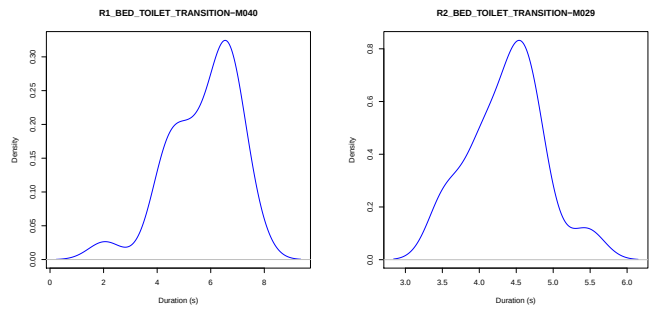


(a) $R1 - D003$ (b) $R2 - D003$

Table 5.7 contains an extract of the results of one of the simulations carried out for DS_3. In this example, all frequencies of activation from the reference data-set for the sensors within the activities *R1-Housekeeping* and *R2-Housekeeping* (column *frequency*) lie within the 0.05-confidence interval (columns *low C.I. (%)* and *upper C.I. (%)*). On the other hand, for 6 sensors, the duration of activation (column *duration*) from the reference data-set does not lay within the 0.05-confidence interval (columns *low C.I. (s)* and *upper C.I. (s)*).

Overall, for DS_3, 99 % of the reference values for frequencies of activations lie within the 0.05-confidence interval, while for 88 % of the reference values for the duration of activations lie within the 0.05-confidence interval.

Figure 5.7: Distribution for the duration of activation of two sensors in DS_3 (multiple-user scenario). (Source : own research)



(a) $R1 - D_{003}$ (b) $R2 - D_{003}$

Table 5.7: Internal Consistency – Example of the results for one of the executions on DS_3 (multiple-user scenario). Lower C.I. refers to Lower limit of the confidence interval and Upper C.I. to Upper limit of the confidence interval. (Source : own research)

activity	sensor name	frequency (%)	lower C.I. (%)	upper C.I. (%)	similar frequency	duration (s)	lower C.I. (s)	upper C.I. (s)	similar duration
R1-BED-TOILET-TRANSITION	D004	3.3	0.9	5.9	✓	100.1	98.2	100.7	✓
R1-BED-TOILET-TRANSITION	M029	6.6	2.2	8.9	✓	3.2	2.9	4.8	✓
R1-BED-TOILET-TRANSITION	M037	7.1	5	12.2	✓	4.3	4.4	5	✗
R1-BED-TOILET-TRANSITION	M038	6.7	2.4	9.4	✓	4.2	4.5	5.1	✗
R1-BED-TOILET-TRANSITION	M039	6.9	3	9.6	✓	4.1	3.8	4.7	✓
R1-BED-TOILET-TRANSITION	M040	6.9	6	14.4	✓	5.9	4.8	6.9	✓
R1-BED-TOILET-TRANSITION	M041	9.4	9.3	22	✓	8.2	6.5	10.5	✓
R1-BED-TOILET-TRANSITION	M043	6.8	3.5	12.2	✓	4.3	3.2	4.4	✓
R1-BED-TOILET-TRANSITION	M044	6.7	3.5	10	✓	6.9	5.4	7.7	✗
R1-BED-TOILET-TRANSITION	M045	6.8	3.7	11.5	✓	4.4	3.8	4.6	✓
R1-BED-TOILET-TRANSITION	M046	10.2	5.2	14.5	✓	6.4	3.4	9.5	✓
R1-BED-TOILET-TRANSITION	M047	6.9	2.8	11.1	✓	12.7	12.2	14	✓
R1-BED-TOILET-TRANSITION	M050	10.7	4.9	13.5	✓	3.6	2.7	4.8	✓
R2-BED-TOILET-TRANSITION	D003	38.7	33.2	54	✓	226.5	224.7	227.5	✓
R2-BED-TOILET-TRANSITION	D005	2.4	1.8	4.5	✓	18.1	17.8	20	✓
R2-BED-TOILET-TRANSITION	M029	2.5	0.2	3.5	✓	4.5	4	4.8	✓
R2-BED-TOILET-TRANSITION	M030	5	1.9	8.6	✓	6.3	6.3	6.7	✗
R2-BED-TOILET-TRANSITION	M032	4.3	4.5	13.4	✓	5.5	4.6	7.7	✓
R2-BED-TOILET-TRANSITION	M033	2.6	0.3	4.5	✓	3.2	0.4	4.7	✓
R2-BED-TOILET-TRANSITION	M034	2.6	0.3	3.5	✓	1.1	1.8	4	✗
R2-BED-TOILET-TRANSITION	M035	2.6	0.6	5.1	✓	2	0.3	4.3	✓
R2-BED-TOILET-TRANSITION	M036	2.5	1	6.4	✓	8.6	7.4	13.2	✓
R2-BED-TOILET-TRANSITION	M037	4.8	1.6	7	✓	11.3	10.2	12.2	✓
R2-BED-TOILET-TRANSITION	M038	5.1	3.4	13.3	✓	8.4	9	15.5	✗
R2-BED-TOILET-TRANSITION	M039	10.2	6.8	14.9	✓	5.1	3.5	6.8	✓
R2-BED-TOILET-TRANSITION	M040	7.3	3.8	12.3	✓	4.4	3.1	5.4	✓
R2-BED-TOILET-TRANSITION	M041	10.2	6.7	14.8	✓	6.2	5.8	9.7	✓
R2-BED-TOILET-TRANSITION	M046	25	14.8	28.9	✓	5.8	3.9	7.7	✓
R2-BED-TOILET-TRANSITION	M047	9.7	5.5	15.4	✓	5.4	4.1	7.1	✓

5.3.1 Coherence with real measurements

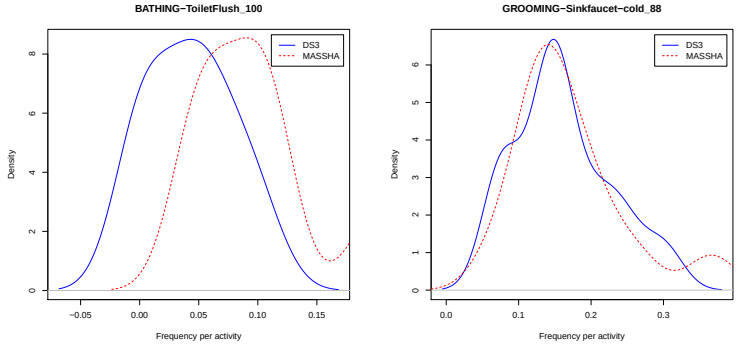
Regarding the *coherence with real measurements*, the objective is to asses whether the proposed methodology is able to create data-sets that are similar to those recorded in real conditions. To this end, I estimated the duration, frequency, and hourly execution of daily activities in order to build a human behaviour model (see Section 5.1.1) for the single-user and multiple-user scenarios. The Geosimulation was then configured to simulate the execution of activities for a 14-day period in both cases.

Single-user scenario

Table 5.8 shows a comparison between the frequency and duration of activation for sensors within the activities $Grooming$, $PreparingBeverage$, $PreparingBreakfast$, and $PreparingLunch$ for the Geosimulation and DS_1. Figure 5.8 compares the distributions for the *frequency* of activation of two sensors between the Geosimulation and DS_1. Figure 5.9 compares the distributions for the *duration* of activation of two sensors for the same data-sets. As can be seen, the distributions for the real and simulated data-sets are similar, meaning that the proposed simulation is able to replicate the original single-user data-set in terms of frequencies and durations of activation.

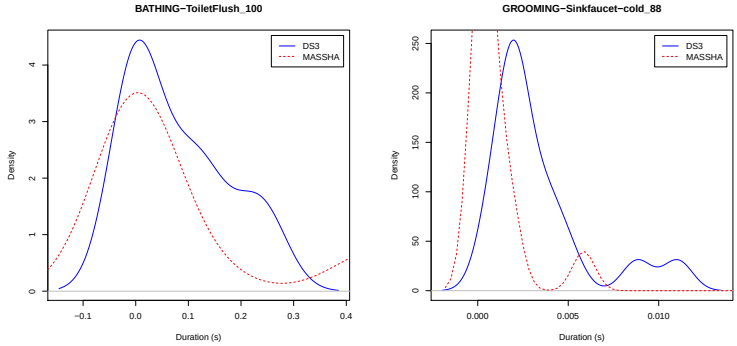
Finally, Table 5.8 shows the results of the similarity test for the frequency and duration for a extract of the sensors and activities taking as *region of similarity* a 15 % of distance (see Section 5.1.4 for details).

Figure 5.8: Distribution for the frequency of activation for two sensors in the Geosimulation and DS_1 (single-user scenario). (Source : own research)



(a) $ToiletFlush_{100}$ (b) $Sinkfaucet-cold_{88}$

Figure 5.9: Distribution for the duration of activation for two sensors in the Geosimulation and DS_1 (single-user scenario). (Source : own research)



(a) $ToiletFlush_{100}$ (b) $Sinkfaucet-cold_{88}$

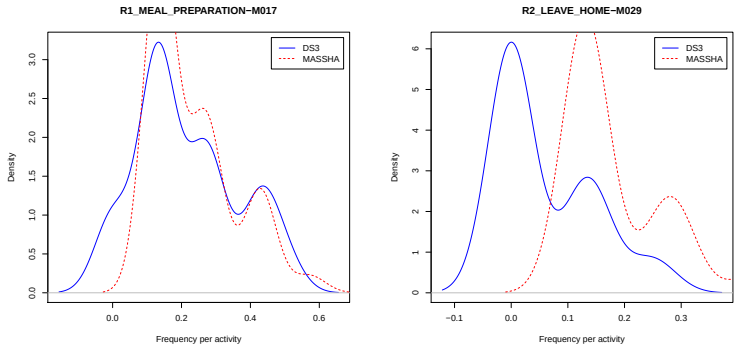
Table 5.8: Coherence with real measurements – Extract of the results of the comparison Geosimulation vs DS_1 (single-user scenario). W refers to Reference data-set and M to Geosimulated data-set. (Source : own research)

activity	sensor name	mean frequency W (%)	mean frequency M (%)	similar frequency	mean duration W (s)	mean duration M (s)	similar duration
GROOMING	Cabinet67	8.68	9.13	✓	6.29	0.08	✗
GROOMING	Door130	10.32	3.06	✗	0.19	0.12	✓
GROOMING	ExhaustFan96	6.84	2.25	✓	0.12	0.45	✓
GROOMING	Jewelrybox139	5.81	3.21	✓	0.03	0.45	✓
GROOMING	Lightswitch101	7.6	5.89	✓	6.46	2.93	✗
GROOMING	Lightswitch120	5.73	1.25	✓	0.03	1.22	✓
GROOMING	Medicinecabinet57	14.54	11.28	✓	63	56.59	✗
GROOMING	Medicinecabinet58	12.45	13.28	✓	6.63	0.8	✗
GROOMING	Sinkfaucet-cold88	16.97	16.12	✓	1.91	0.23	✓
GROOMING	Sinkfaucet-hot68	17.97	19.58	✓	2.09	0.33	✓
GROOMING	ToiletFlush100	8.06	1.73	✓	7.04	3.16	✗
PREPARINGBEVERAGE	Cabinet80	21.5	16.5	✗	4.15	0.14	✓
PREPARINGBEVERAGE	Freezer137	22.14	6.07	✗	18.6	15.29	✗
PREPARINGBEVERAGE	Refrigerator91	24.27	21.25	✗	12.93	0.42	✗
PREPARINGBREAKFAST	Cabinet73	12.81	2.5	✗	9.18	0.02	✗
PREPARINGBREAKFAST	Drawer84	11.61	7.5	✓	1.36	0.02	✓
PREPARINGBREAKFAST	Freezer137	9.7	2.92	✗	21.7	4.96	✗
PREPARINGBREAKFAST	Refrigerator91	11.71	6.56	✗	3.53	0.02	✓
PREPARINGBREAKFAST	Toaster131	27.06	10.63	✗	4.88	5.65	✓
PREPARINGDINNER	Cabinet55	7.02	5	✓	1.35	0.06	✓
PREPARINGDINNER	Cabinet80	6.62	7.5	✓	1.34	0.02	✓
PREPARINGDINNER	Drawer84	7.51	3.33	✓	0.35	0.02	✓
PREPARINGDINNER	Freezer137	18.3	16.43	✗	18.75	4.24	✗
PREPARINGDINNER	Refrigerator91	18.98	15.63	✗	3.26	0.21	✓
PREPARINGLUNCH	Cabinet53	6.17	4.77	✓	2.38	0.42	✓
PREPARINGLUNCH	Cabinet55	6.82	6.14	✓	1.55	0.75	✓
PREPARINGLUNCH	Cabinet59	4.76	3.33	✓	0.36	0.06	✓
PREPARINGLUNCH	Cabinet72	7.52	3.33	✗	0.15	0.02	✓
PREPARINGLUNCH	Cabinet80	5.75	0.71	✓	1.26	0.02	✓
PREPARINGLUNCH	Dishwasher70	5.37	2.86	✓	12.57	11.36	✗

Multiple-user scenario

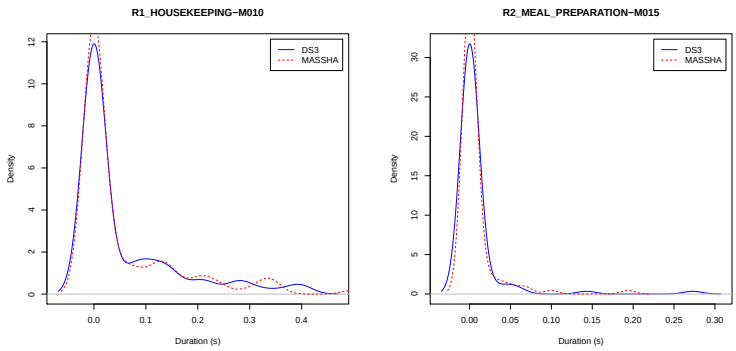
Table 5.9 shows a comparison between the frequency and duration of activation for sensors within the activities $R1\_Eating$, $R2\_LeaveHome$ for the Geosimulation and DS_3. Figure 5.10 compares the distributions for the *frequency* of activation of two sensors between the Geosimulation and DS_3. Figure 5.9 compares the distributions for the *duration* of activation of two sensors for the same data-sets. As can be seen, the distributions for the real and simulated data-sets are similar, meaning that the proposed simulation is able to replicate the original multiple-user data-set in terms of frequencies and durations of activation.

Figure 5.10: Distribution for the frequency of activation for two sensors in the Geosimulation and DS_3 (multiple-user scenario). (Source : own research)



(a) $R1 - M_{017}$ (b) $R2 - M_{029}$

Figure 5.11: Distribution for the duration of activation for two sensors in the Geosimulation and DS_3 (multiple-user scenario). (Source : own research)



(a) $R1 - M_{010}$ (b) $R2 - M_{015}$

Table 5.9: Coherence with real measurements – Extract of the results of the comparison Geosimulation vs DS_3 (multiple-user scenario). W refers to Reference data-set and M to Geosimulated data-set. (Source : own research)

activity	sensor name	mean frequency W (%)	mean frequency M (%)	similar frequency	mean duration W (s)	mean duration M (s)	similar duration
R1-EATING	M013	13.84	1.56	✓	4.24	1.91	✓
R1-EATING	M014	17.26	5.57	✓	3.68	3.39	✓
R1-EATING	M034	17.93	12.41	✓	13.14	10.6	✓
R1-EATING	M035	29.49	26.46	✗	6.26	3.83	✓
R1-LEAVE-HOME	D001	21.51	11.43	✓	10.27	5.5	✓
R1-LEAVE-HOME	M024	24.82	14.94	✓	13.89	8.88	✓
R1-LEAVE-HOME	M025	21.8	14.96	✓	9.44	6.32	✓
R1-LEAVE-HOME	M026	22.54	13.33	✓	5.12	2.93	✓
R1-LEAVE-HOME	M027	20.66	18.03	✗	3.76	3.78	✓
R1-LEAVE-HOME	M028	22.93	14.22	✓	5.42	3.66	✓
R1-LEAVE-HOME	M043	22.31	12.59	✓	2.09	1.35	✓
R2-EATING	M006	26.31	22.76	✗	7.38	7.45	✓
R2-EATING	M007	21.61	12.15	✓	6.65	3.64	✓
R2-EATING	M008	18.87	14.38	✗	3.02	2.31	✓
R2-EATING	M009	21.22	13.87	✓	4.86	3.68	✓
R2-EATING	M015	16.64	6.43	✓	4.23	1.46	✓
R2-EATING	M016	19.51	8.02	✓	5.27	2.18	✓
R2-EATING	M017	21.34	13.9	✓	14.1	8.6	✓
R2-EATING	M018	14.73	1.19	✓	3.25	0.16	✓
R2-EATING	M046	14.96	4.91	✓	0.39	0.12	✗
R2-EATING	M051	13.78	1.79	✓	0.85	0.18	✓
R2-LEAVE-HOME	D001	17.01	10.24	✓	17.01	10.24	✓
R2-LEAVE-HOME	M024	18.11	9.67	✓	18.11	9.67	✓
R2-LEAVE-HOME	M025	16.69	7.01	✓	16.69	7.01	✓
R2-LEAVE-HOME	M026	16.83	9.09	✓	16.83	9.09	✓
R2-LEAVE-HOME	M027	16.33	7.79	✓	16.33	7.79	✓
R2-LEAVE-HOME	M028	16.55	9.1	✓	16.55	9.1	✓
R2-LEAVE-HOME	M029	18.1	6.03	✓	18.1	6.03	✓
R2-LEAVE-HOME	M030	16.96	10.65	✗	16.96	10.65	✗
R2-LEAVE-HOME	M032	16.57	9.55	✗	16.57	9.55	✗
R2-LEAVE-HOME	M035	23.69	17.87	✗	23.69	17.87	✗
R2-LEAVE-HOME	M036	16.63	4.99	✓	16.63	4.99	✓

5.4 Discussion

5.4.1 Overall assessment

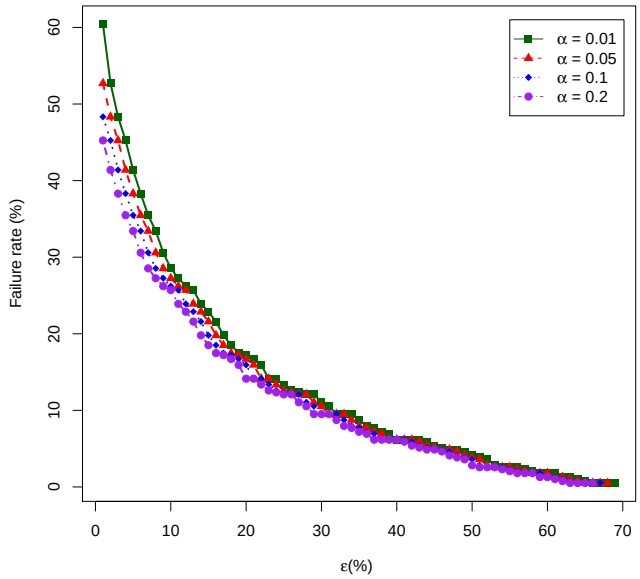
Results validate that the Geosimulation is able to generate activity data-sets that resemble those recorded and annotated in a smart home, regardless of the monitoring approach and the number of occupants in the house. In this sense, the similarity between the reference and the simulated data-sets is given by the resemblance of the *frequency* and *duration* of activation of the sensors depending on the activity being performed at each moment. The particular characteristics and significance of the results obtained for the single-user and multiple-user scenarios are discussed next.

5.4.2 Single-user scenario

Figure 5.12 synthesises the performance of the Geosimulation regarding the consistency with real measurements. In the general case, identifying the acceptable threshold for the region of similarity is not feasible. The specific value may vary depending on the research focus, the variability of human behaviour and the monitoring time. Thus, every practitioner should check whether the results shown in Figure 5.13 satisfy their requirements, before deciding whether they use the Geosimulation for their experiments. However, taking into account the behaviour variability I observed in DS_1 and DS_2, as well as the low number of monitored days (14 days), It is believed the Geosimulation's performance is good. For a region of similarity of 20%, the failure rate of produced sensor activations and their duration is around 15%. So the similarity between the simulated data-set and the real one is around 85% for a reasonable region of similarity, under the conditions described. In scenarios where activity models can be acquired more accurately, the Geosimulation's performance is expected to be even better.

Since the annotation of DS_1 and DS_2 included detailed information regarding sensor activation per hour, I decided to perform an additional test to verify whether the Geosimulation was able to reproduce this behaviour as well. Table 5.6 shows, for every sensor in DS_1 and for every hour of the day, whether the real hourly activations

Figure 5.12: Distribution of the amount of not similar sensors depending on the length of the *region of similarity* (epsilon) and *significance level* (alpha) for DS_1 (single-user). (Source : own research)



of sensors matched those simulated. For example, for sensor $Burner_{106}$, the Geosimulation coincided with its activations for hours h_1, h_6, h_{11}, h_{14}, h_{17}, h_{20}, h_{22} and h_{23} while did not coincide with its activations for hours h_2, h_7, h_{12} and h_{15}. Overall, the hourly activations of the real data-set and that produced by the Geosimulation seem to be quite different. The main cause is that the original data-set contains annotations and monitoring for a short period of time (only 14 days). This causes some of the activities to be wrongly annotated and the definition of actions within activities to be somewhat sloppy. Moreover, the subjects did not follow any consistent patterns neither throughout the day nor the week. Once again, this issue can be overcome by increasing the monitoring period in order to achieve a better definition of behaviour patterns and a more consistent outline of activities and related activation of sensors.

Please note that, in the case of the duration of activation, the amount of variance recorded from real environment (due to errors in the annotation methods). Moreover, the Geosimulation is able to closely recreate data-sets recorded by humans since the amount of sensors without similarity is low even with a small region of similarity. In fact, to get a 15% of disagreement, the *region of similarity* should only be of 21 %. Figure 5.12 shows the distribution of the amount of sensors that are not similar depending on the length of the *region of similarity* (epsilon) and the significance level (alpha). As expected, an increment in both values produces a reduction of the failure rate. Moreover, it can be observed that the failure rate is more sensitive to changes on the region of similarity than to the significance level.

5.4.3 **Multiple-user scenario**

Figure 5.13 synthesises the performance of the Geosimulation regarding the consistency with real measurements in a multiple-user scenario. It can be observed how the failure rate for sensor activations is generally lower than in Figure 5.12, which describes the single-user scenario. For example, given a region of similarity of 10% the Geosimulation achieves a failure rate of around 15%, taking into account the different significance levels depicted in the graphic. Please, notice that to achieve a similar failure rate in the single-user scenario, the region of similarity was around 20%, which shows that the performance of the simulator is better in the multiple-user scenario.

In general, it can be argued that it is harder to simulate two persons instead of one. However, I obtained better results in the multiple-user scenario. This can be explained by three main factors:

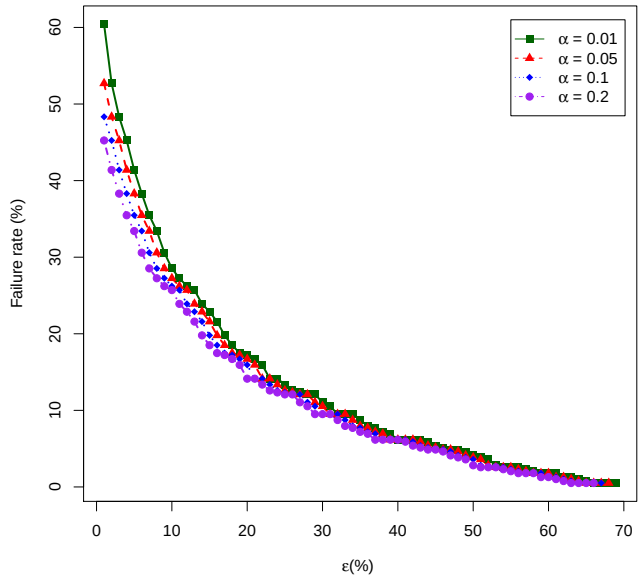
- The monitoring time: in the case of DS_3 I have around 9 months of sensor activations and annotated activities, whereas for DS_1 and DS_2 I only have 14 days. A longer monitoring period allows us obtaining more accurate behaviour models, which produces better simulations.
- Regularity of behaviours: although DS_3 presents two occupants in the same house, their behaviour is more regular than DS_1 and DS_2. Of course, this is also

related to the monitoring time, since longer periods usually show better the behaviour patterns. As the proposed methodology relies on behaviour models, more regular behaviours tend to make the simulation easier.

- The monitoring approach: DS_3 uses motion sensors extensively to monitor the occupants' activities. As the Geosimulation models spatial and temporal restrictions realistically, I can specify typical trajectories when performing activities. I found that location- and motion-based monitoring approaches can be simulated accurately.

Following the approach to validate the single-user scenario, I decided to perform an additional test to verify whether the Geosimulation was able to reproduce the hourly activations of sensors. Table 5.7 shows, for every sensor in DS_3 and for every hour of the day, whether the real hourly activations of sensors matched those simulated. For example, for sensor $M001$, the Geosimulation coincided with its activations for hours h_7, h_{15}, h_{16}, h_{18}, h_{20}, h_{21}, and h_{22}, while did not coincide with its activations for hours h_9, h_{13}, and h_{14}. The overall activations of the real data-set and that produced by the Geosimulation, seem to be quite different. However, in contrast with the results for the single-user scenario shown in Table 5.6, it seems that the Geosimulation was able to perform better with the multiple-user data-set. These results confirm again the importance of longer monitoring times, regular behaviours and the monitoring approach for activities.

Figure 5.13: Distribution of the amount of not similar sensors depending on the length of the *region of similarity* (epsilon) and *significance level* (alpha) for DS_3 (multiple-user). (Source : own research)



5.5 Physical model results

The FIWARE-LAB cloud environment provided a working production instance of the FI-WARE stack without any Service Level Agreement. As in any other cloud environment, the virtual instance is non-shared (namely, it can be guaranteed that no one else is using that instance) but we share virtual machines with several other users (it cannot be guaranteed the amount of bandwidth available nor the overall load of the physical server). In order to asses the different middleware architectures, we defined two Key Performance Indicators (KPI) of interest:

Bandwidth: number of bytes transmitted per unit of time. In this particular example, I recorded the number of bytes per message without including the overheads

related to the specific protocols used to transport the information. The objective of this chapter is to estimate the real bandwidth needs in a real deployment without assuming a simultaneity factor. Note that this value is specially relevant in the Smart City scenario, as the deployment is traditionally performed over SIGFOX or LoRa networks that have limited bandwidth (Adelantado et al., 2016).

Number of Messages Lost: number of messages emitted by GeoWorldSim that are not processed by the Context Broker because they may get lost during the transmission. This value gives information about the quality of the service provided by the architecture. In the experiments, none of the transmitted messages were lost due to the fact that the Context Broker uses an asynchronous database storage, and therefore, during high load peaks, message insertion gets asynchronously scheduled. This is a good indicator of the reliability of the solution.

In order to accelerate the procurement of results and balance both the length of the configuration file and RAM consumption, the simulation time was accelerated. We can define that 1 second of the simulation corresponds to n second in terms of real time. n have been taken accordingly to ease the deployment of the simulation. Modification in n has a direct impact in the relation between the measured KPIs and the real KPIs (i.e. if $n = 1$ have been used). In this sense, an speedup factor of n would be equivalent to:

- multiply by n the number of houses
- divide by n the bandwidth

Several city sizes were tested for each of the scenarios defined in **??**, specifically, Village (250 houses), Small City (2500 houses), Medium City (25 000 houses), Large City (250 000 houses) and Megalopolis (2 500 000 houses). Each house accommodates two persons that interact with 18 smart devices while performing their daily duties. In addition, the scenarios were provided with (5 × #number of houses) streetlights and (1 × #number of houses) recycling containers.

The simulation has been carried out for 6 hours, the busiest of the day, that is 3 in the morning and 3 in the evening:

Morning: This portion of the simulation covers the time ranging from the moment the citizens wake up to the moment they go to work. The appliances activated are those related to the activities that involve getting ready for going outside (showering, dressing, grooming), and preparing breakfast. The number of appliances used is rather low but all of them are used almost at the same time.

Evening: This portion of the simulation covers the period of time between having dinner and going to bed. This period includes the prime time. The appliances activated range from the ones typically used in the kitchen to the ones used for entertainment. In this time period, the number of appliances used is high and diverse but their activation is more spread over time.

Since each simulation run is done over a long period of time, we consider that there is no need to repeat the experimentation due to the fact that all the possible variability that may appear will probably be already captured. A video of a complete simulation can be found in `http://bit.ly/2fbX54p`.

For each simulation we register two logs: the timestamp and types of message sent by GeoWorldSim and the equivalent log from the FI-WARE Context Broker so as to easily extract the KPIs values. For example, Figure 5.14 shows a graphical representation of the number of messages per device transmitted during the simulation time. The bandwidth is measured by multiplying the number of messages by their length and dividing the result by the unit of time.

Figure 5.15 shows several boxplots (Benjamini, 1988) depicting the bandwidth distribution used per number of houses in the Smart City scenario. Please note that this is a log-log plot. As can be seen, the bandwidth grows and its saturation takes place at the *Medium City* level, with a saturation bandwidth around 1 MB/s. Take into consideration that this scenario is the only one where the amount of messages allows to perform a simulation of the Megalopolis. Clearly, even though we were able to perform this simulation, the computational needs superseded the bandwidth restrictions of this case.

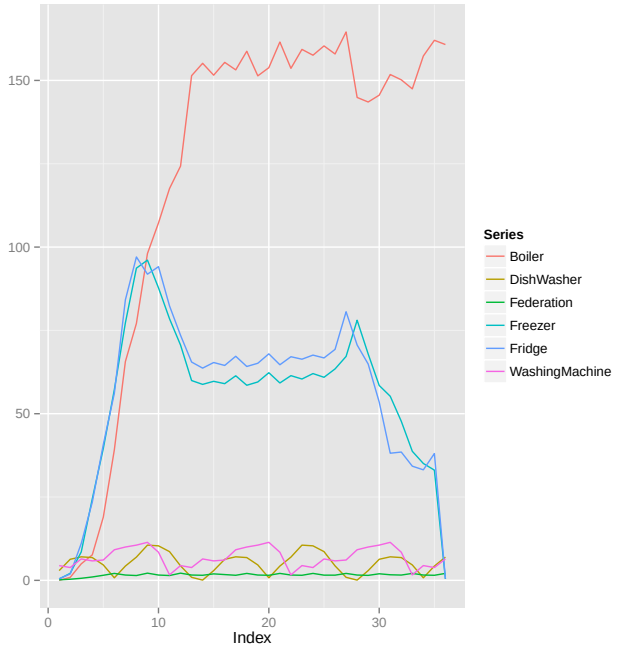


Figure 5.14: Evolution of the number and type of messages per time in a simulation. (Source : own research)

The IoT Jungle represents the most problematic scenario. Figure 5.16 shows, as in the previous case, several boxplots describing the distribution of the bandwidth used per size of the city. As can be seen, the saturation is reached under the same case *Medium City*, but it uses 10 times more bandwidth than the Smart City scenario. However, in this case, it seems that the bandwidth demands of the *Large City* have produces a change of phase and the bandwidth collapses (Al-Bahadili, 2012).

Federation architectures need to have several Context Brokers operating simultaneously. Only access to a single instance of a Context Broker is available, therefore we cannot directly asses this architecture. However, the simulation can be performed in two steps. First, the behaviour is simulated on a node on the federation (a house) and register all the messages that should have been sent to the federation. Then this

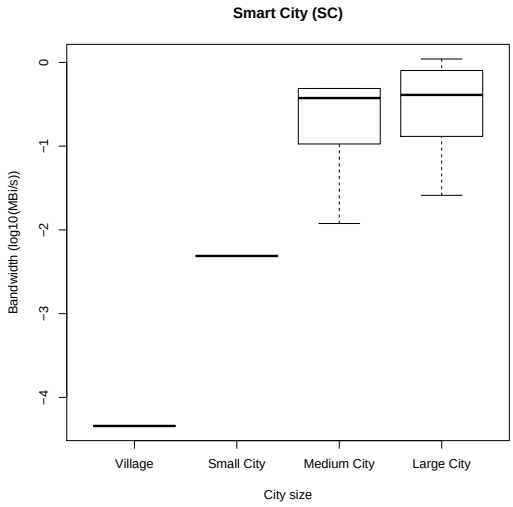


Figure 5.15: Bandwidth distribution per number of houses in the Smart City scenario. (Source : own research)

process is repeated for every node. Finally, the message record is replayed from the nodes to the centralised Context Broker. Figure 5.17 shows the bandwidth used per number of houses in the Ubiquitous World scenario. In this case the saturation takes place in the *Large City* case, in contrast with the previous two scenarios.

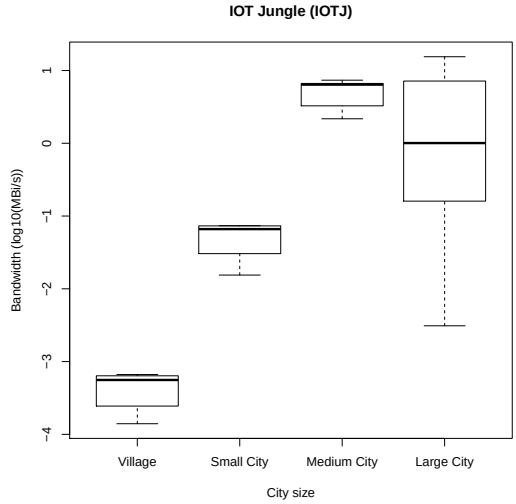


Figure 5.16: Bandwidth used per number of houses in the IoT Jungle scenario. (Source : own research)

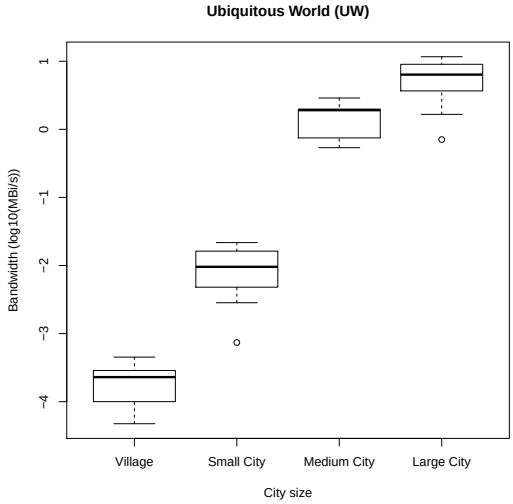


Figure 5.17: Bandwidth used per number of houses in the Ubiquitous World scenario. (Source : own research)

CHAPTER

6

Use-case 2 - Green Travelling

Around a quarter of the greenhouse gas (GHG) emissions produced within the European Union (EU) is direct responsibility of the transportation sector, making it the second largest GHG emitter after the energy sector. The transportation sector is typically divided in five modes, namely road, rail, maritime, aviation and pipelines. Out of these modes, road transportation is considered the largest source of GHGs as it stands for one-fifth of the total CO2 emissions of the EU (COM-EU-Commission et al., 2007). In fact, far from diminishing, these emissions are still growing.

The majority of GHG emissions from road transportation are derived from the combustion of petroleum based-products, such as gasoline and diesel, in internal combustion engines. The largest sources of transportation-related GHG emissions include passenger cars and light-duty trucks, such as sport utility vehicles, pickup trucks, and minivans. The remainder of the GHG emissions comes from other means

of transportation, including freight trucks, commercial aircraft, ships, boats, and trains as well as pipelines and lubricants.

The EU has so far put a range of policies in place aiming at lowering emissions in the transportation sector, including strategies to reduce emissions from vehicles, minimise GHG intensity related to fuels or require public authorities to take account of lifetime energy use and CO2 procuring vehicles (Skinner, 2014). However, implementing these policies without a prior analysis can caught the authorities off guard and backfire the efforts to reduce environmental and social impact. As occurred with the Norwegian incentives model (Holtsmark and Skonhoft, 2014), allowing electric vehicles to use public infrastructure has resulted in saturating bus lanes where time lost by thousands of bus passengers became far greater than that gained by a few electric car drivers.

Therefore, the precise quantification of the impact resultant from sustainable transport policies is essential in order to take the right decisions and not creating a future problem. One of the most common approaches used for this purpose is traffic modeling, which consists in emulating an entire road network with vehicles travelling through in order to be used in decision support tools. These tools are typically employed by infrastructure managers and public administrations to evaluate different parameterizable scenarios (e.g. with electric vehicles usage quotes, electric vehicles features, incentive policies, etc.) according to several costs, as well as social and environmental impacts.

Traditional approaches that calculate the impact of the travelling sector consist on creating a specific, whether fixed or approximated, amount of vehicles that represent a vehicle fleet, schedule their itineraries throughout the city, estimate the global impact derived from the configuration of a single scenario, and compare the results with those obtained from the simulation of other scenarios.

Multi-agent systems have been an active area of research in the last three decades for transportation modelling. The literature shows a great amount of research concerning traffic simulation through agent-based systems. The most important projects in this area are detailed as follows.

Sumo (Krajzewicz et al., 2012): An open-source, microscopic and multi-modal traffic simulation package designed to handle large road networks and establish a common test-bed for algorithms and models from traffic research. It facilitates interoperability with external applications during run time, using TraCI (Traffic Control Interface). SUMO comes with a mechanism to import cartographic material and automatically generate an input for traffic simulation taking data directly from OpenStreetMap. The main drawback of SUMO is having to explicitly define by hand multi-modal route steps for each citizen instead of the simulation being able to calculate them according to the existing available public transport vehicles.

MATSim (Horni et al., 2016): An open-source, agent-based transportation simulation that is able to simulate large-scale scenarios. Currently, MATSim offers a framework for demand-modelling, agent-based mobility-simulation (traffic flow simulation), re-planning, a controller to iteratively run simulations as well as methods to analyse the output generated by the modules. Originally MatSim only considers simulations for private car traffic, trimming its capabilities to answer sophisticated questions posed to advanced models that include several types of vehicles. However, in later versions the functionality was extended to all kinds of public transports, including pedestrians and cyclists.

Vissim (Fellendorf, 1994): A microscopic, time step and behaviour based simulation model developed to analyse the full range of functionally classified roadways and public transportation operations. It models cars with a high-fidelity model based on the Wiedemann-Model. and allows to very accurately create traffic simulations defining the amount of vehicles and types that travel through its network. It is very recommended for highly realistic studies but not suitable for large scale simulation studies.

PRIMES-TREMOVE (Siskos and Capros, 2014): An economic model for transportation simulation that combines modelling of micro-economic behaviour concerning distribution of mobility and vehicle choices. It contains a transport demand module based on decision trees emulating the decision process of dif-

ferent profiles choosing their best transport method according to their income constraints.

These methodologies are relatively easy to follow and have have many proven success stories but have a fundamental problem: they focus primarily on the calculation and estimation of routes and congestion avoidance, but fail to consider the social behaviour which influences the decision of using one transport method or the other. Simulations need to consider this previous step to include how the social behaviour affects the output of the evaluation of a scenario. This implies modelling social preferences when travelling such as time cost, comfort, monetary cost or environment friendly awareness.

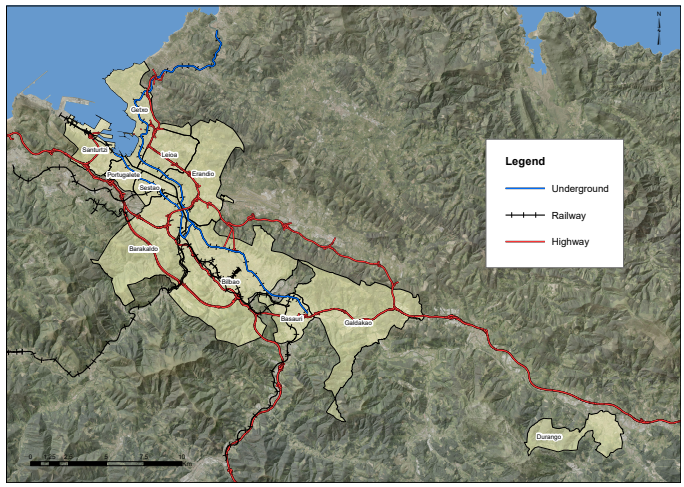
Actual platforms for road simulation do not cover these needs, either due to the impossibility to parameterise the initial system configuration according to social variables, or due to the distribution of such modules as additional commercial packages. *In order to overcome these issues, this chapter designs and implements a methodology on GeoWorldSim.* Everyday commutes follow the conventional four-step structure consisting on trip generation, destination choice, transport mode choice and route choice. In order to achieve a simulation that faithfully represents this reality, the following sections describe the structure of the Geosimulations.

6.1 Modelling social behaviour

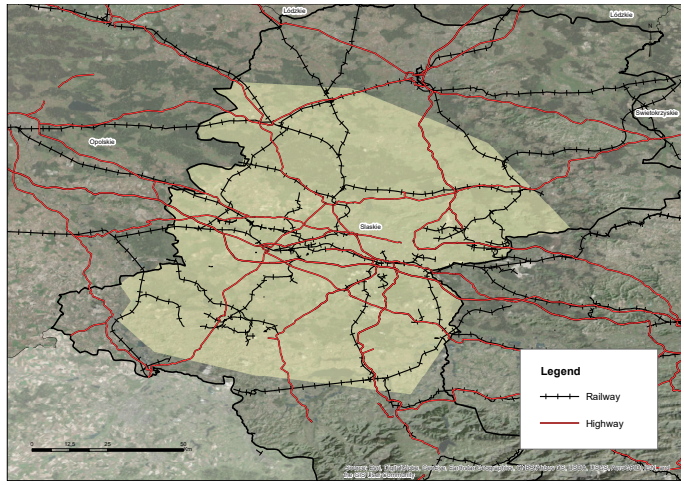
In order to generate the information to model everyday decision making when travelling, I created a Geosimulation where the main characters were citizens commuting from their houses to working or studying place. Each citizen agent had a set of personal preferences that directly influenced the transport choice for travelling. For the generation and distribution of the trips for the experimentation, two data sets were used. On the one hand, Biscay's commute data (depicted in Figure 6.1a and Figure 6.2a) was extracted from the census performed every 10 years by the Spanish National Statistics Institute (Instituto Nacional de Estadística, 2011). This data set covers the region of Biscay, in the Basque Country, with an area of 2200 km^2. The census also details the points of origin and destination of travelling for the most

populated 11 municipalities of the region, with a total population of about 1,100,000 citizens. This data set will be used to train the different models. On the other hand, Upper Silesia Conurbation's commutes (depicted in Figure 6.1b and Figure 6.2b) were extracted from the green travelling project's surveys (Green Travelling Consortium, 2013). The data-set covers 19 cities of the Upper Silesian Conurbation with a population of about 2 300 000 citizens and 4 700 000 commutes every day. This data set was used to validate the results.

When the data set is brought to life by the simulation, each citizen agent registers its point of origin, point of destination, departure time, and the itinerary for the five transport modes. The platform preprocesses the input data to geolocate each citizen agent on the environment through a weighted distribution. This process assigns the origin of each citizen agent to household buildings in the municipality and the destination to available amenities, commercial, or industrial buildings. The weighted distributions are based on the buildings' size and levels, i.e., the bigger and the taller the building, the higher the probability of assigning a citizen agent to it. Once located, citizens will calculate the five itineraries for *CAR*, *MOTORCYCLE*, *TRANSIT*, *BICYCLE*, and *WALK*. For each itinerary, the main characteristics of every choice (see Section 6.1.1 for details) will be stored in order to build their decision process. Some itineraries might be null due to a lack of public transport stop nearby or an inability to calculate the route.

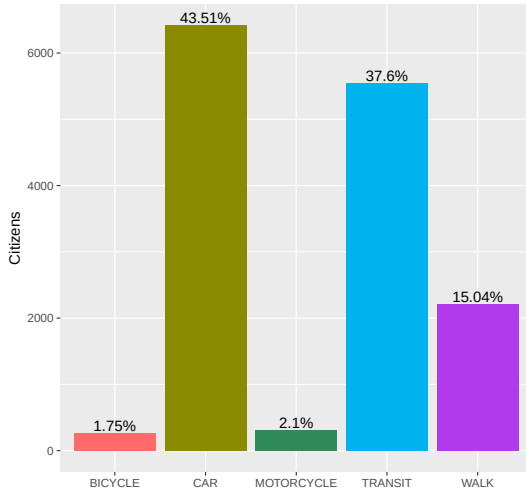


(a) Biscay

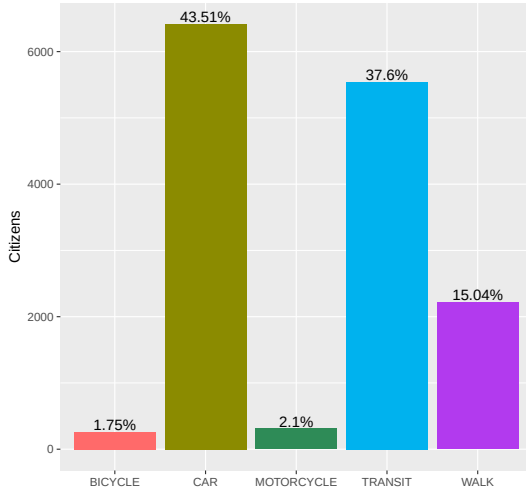


(b) Silesia

Figure 6.1: Area and transport links of Biscay (Spain) and Silesia (Poland) data sets. (Source: own research generated from real boundaries and transport networks)



(a) Biscay



(b) Silesia

Figure 6.2: Modal splits for the Biscay and Silesia data sets. (Source: own research from the datasources of (Instituto Nacional de Estadística, 2011; Green Travelling Consortium, 2013))

6.1.1 Transport Mode Choice

There are many features that determine the modal split of a territory. More than 45 years ago, the authors of (Domencich and McFadden, 1975) identified the main features that influence commuters' choices, and since then other important characteristics have been added to the list (Chiu Chuen et al., 2014). In spite of using different techniques, researchers agree that itinerary dependent variables can be grouped into four main categories that represent the physical and factual features of a trip (Clauss and Döppe, 2016): duration, price, length, and environmental impact. Table 6.1 describes these categories and shows how they are usually further divided in other second-level features. Nonetheless, these four variables do not cover some other features listed in the literature, which are directly linked to the traveller's background. These background variables, presented in Table 6.2 do not appear in the census nor in the surveys and are therefore beyond the scope of this study (see Section 6.4 for details).

Missing values have been the main source of problem. In these data sets, missing values appear as NaN (not a number), representing that the itinerary is undefined. There are valid reasons to have missing values in the data set (for example, it is not possible to travel from one place to other using public transport), but a closer look reveals that the missing values are randomly distributed and do not follow any logical pattern (for example, itineraries that can be performed by *MOTORCYCLE* and not by *CAR*). This suggests that missing values originated from the routing algorithm time-out errors due to the inability to calculate complex routes. For this reason, all NaN values were replaced with an arbitrary number that was greater than any of the real measured values (in this case, all NaNs were replaced by the maximum value in the data set).

The rest of the variables do not present any obvious problems. In fact, they clearly show an expected distribution. For example, Figure 6.3 shows the length of the trips made with BICYCLE. As expected, there is a point where the trips using *BICYCLE* are too costly and fall sharply. Another expected result is, for example, the distribution of time expended in *CAR* trips. As seen in the left panel of Figure 6.3, the distribution

almost follows a χ^2 distribution. As citizens tend to be normally geographically distributed (Pijoan et al., 2015a), the distance traveled follows a χ^2 distribution, as does the time spent.

The influence of these variables on transport choice was modelled through eight modelling techniques: k-nearest neighbours (KNNs) (Altman, 1992), multinomial logit (M) (Lemeshow et al., 2013), support vector machines (SVMs) (Cristianini and Shawe-Taylor, 2000), neural networks (NNs) (Schalkoff, 1997), naïve Bayes (B) (Lewis, 1998), fuzzy logic (CE) (Mendel, 2001), expert knowledge (EK) (Souche, 2010), and random search (RA).

Table 6.1: Itinerary features used for transport choice. (Source: Adapted from (Clauss and Döppe, 2016))

Feature	Description	Encompasses
Duration	Time necessary (in seconds) to complete the itinerary from the departure point until arriving at the destination.	Closeness to public transport stop and parking spaces. Frequency and reliability of public transport. In-journey travelling time. Traffic congestion. Transfer waiting time between two means of transport. Waiting environment. Poor waiting time.
Price	Monetary cost (in euros) of the itinerary including public transport tickets, parking fares and private vehicle costs.	Public transport fares. Private vehicle purchase, maintenance, and fuel costs. Tolls and congestion charges. Economic incentives.
Length	Distance of the itinerary (in kilometres). For long- and medium-length travels, this is not a determinant factor. This is only when walking or cycling are an option.	Walking distance. Cycling distance.
Environmental impact	Contribution to climate change (measured in CO_2 emissions). Regarding short distance and itineraries, where transports compete in similar conditions, environmental awareness can be the trigger that determines user decisions.	Environmental impact of the trip.

Table 6.2: Itinerary features are not present in the data sets and are beyond the scope of the study. (Source: own research, conducted by analysing the features mentioned in (Domencich and McFadden, 1975), which are not present in the data sets.)

Feature	**Description**	**Encompasses**
Profile	Personal demographic, socio-economical, and cultural attributes of the user.	Citizen age Citizen gender Socio-economic profile Cultural profile Family size Trip type
Ownership	Vehicle(s) under the property of the user.	Car ownership Motorcycle ownership Bicycle ownership
Climatology	Variables that may have a direct impact on the physical variables of the trip, i.e., if it rains, the duration of a trip may take longer than usual.	Environment climatology

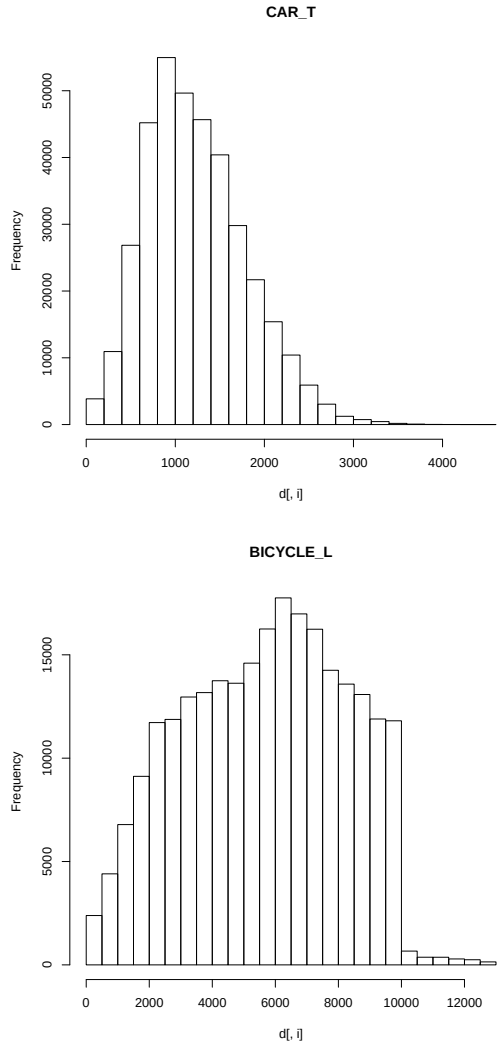


Figure 6.3: Distribution of time (**left**) and length (**right**) for *CAR* and *BICYCLE* transport modes in Silesian Voivodeship. (Source : own research)

Multinomial Logit Models

Following the so-called *latent construct interpretation*, multinomial models work by adjusting a helper function p that takes as input a vector of external factors that affect the choice and gives as output the selected one. For a two class case, the function p takes the form:

$$p(X) := \begin{cases} 1 & \beta X + \varepsilon > 0 \\ 0 & else \end{cases}$$

where X is a vector of the variables considered to make the regression, β is the vector of regressors to be estimated, and ε is a random variable following the logistic (logit) or normal (probit) distribution. Details can be consulted in (Lemeshow et al., 2013).

Support Vector Machines

This technique builds a black-box model that can be tailored for classification and regression problems (Cristianini and Shawe-Taylor, 2000). A more detailed explanation is beyond the scope of this document and readers interested can consult (Cristianini and Shawe-Taylor, 2000) for details. Since the transport choice is a classification problem, the default approach ε-classification was followed. This technique features three hyper-parameters: C controls the penalisation to the errors in the training sample, ε controls the margin of tolerance to errors, and γ controls the bandwidth of the radial function. These hyper-parameters have been optimised using a grid search following the advice given in (Chang and Lin, 2011).

Neural Networks

Technique that creates a black-box model that tries to mimic the behaviour of the natural neurons. This way, a neural network is composed of a set of artificial networks arranged into three layers: an input layer, a hidden layer, and an output layer (see Figure 6.4). The outputs of one layer are connected to the inputs of the subsequent layers, building in this way a network of neurons. Finally, each neuron computes the

following function:

$$y_i := \varphi\left(b_i + \sum_{k=0}^{m} w_{ki} x_k\right)$$

where x is the input vector w, b are the parameters of the model (wights and bias), and $\varphi := \frac{1}{1+e^{-x}}$ denotes the sigmoid function (David, 2006). Further details can be consulted in (Schalkoff, 1997). The hidden layer features several numbers of neurons, and the parameters of the model have been adjusted through back-propagation.

The main hyper-parameter of this model is the number of neurons m in the hidden layer. In this case, a grid search has been followed to optimise this value such that $m = 100$ is the best value.

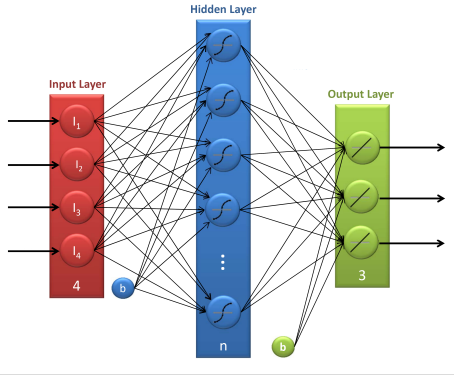


Figure 6.4: Structure of a neural network (NN).

The Naïve Bayes (B) model

This model extends Bayes' theorem to perform classifications. This way, this model estimates a probability $p_c(X)$ of every transport mode given the vector of variables considered to make the prediction X. For this end, the algorithm just needs to calculate an estimation of the probability of X conditional of every transport mode c and then use Bayes' Theorem. The output class will be the one with the greatest probability. Details of this model can be found in (Lewis, 1998).

k-Nearest Neighbours

This method uses the notion of the *closeness* of the input to perform classifications. This way, given a new element y to classify, the algorithm searches the k-nearest elements known in the training set with respect to a given metric (usually the Euclidean) and classifies y as the weighted mean value of the nearest neighbours (in a regression setting) or the most common class among the k-nearest elements (in a classification setting as this one). Using a grid search procedure, it was found that $k = 5$ was the best value for this project.

Fuzzy Logic

All methods presented above are black-box models or are not straightforward in considering new transport modes or new input variables. Fuzzy logic (Mendel, 2001), however, has been shown to achieve high success in the emulation of individual preferences for transport mode choice (Kumar et al., 2013; Tuzkaya and Önüt, 2008), therefore enabling a wider level of parameterisation for each citizen's profile. A well trained fuzzy logic engine and accurately defined rule sets are expected to provide consistent results even in the case of variation in the characteristics of the transport network.

Fuzzy logic engines build discrete choice models through a simple natural language rule-based approach (Berkan and Trubatch, 1997). Classical logic only allows conclusions to be true or false. In contrast, fuzzy logic defines a set of rules that map numeric data into linguistic terms and creates fuzzy sets, indicating the extent to which each term is part of. That is, a variable can have several values (called terms) that can be overlapped and shaped. Thereby, it is possible to model how each feature increases or decreases the likelihood of choosing a given transport mode. Fuzzy logic engines are parameterised though their inputs, outputs, and rules. For this research, the fuzzy logic was built as follows:

Inputs, terms, and membership functions: For the engine to translate from linguistic terms to numbers, inputs are composed of a name that identifies the input, the possible terms or values that the input can take, and a membership func-

tion that limits the extent to which each term can be classified. The inputs of the model presented here are built from the four previously mentioned itinerary features: *DURATION*, *PRICE*, *LENGTH*, and *ENVIRONMENTAL IMPACT*. Each of these inputs feature a fixed set of three terms: *LOW*, *MEDIUM*, and *HIGH*, shaped as triangles. The anchors of these triangles (α, β, γ, and δ), shown in Figure 6.5, are key values that describe how citizens understand these qualitative variables.

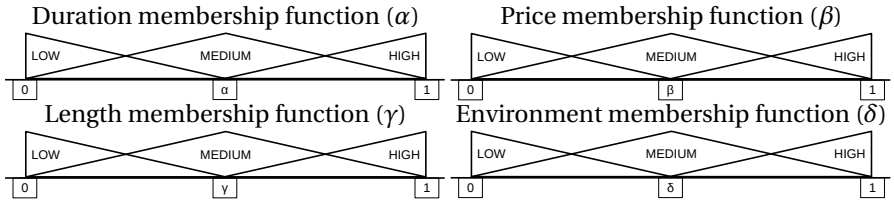


Figure 6.5: Fuzzy engine input membership functions, with the configurable anchors that modify the shape of the terms. (Source: own research for better understanding the input membership functions)

Outputs, terms, and membership functions: Similarly to the inputs, the outputs are composed of a name that identifies the output, the possible terms or values the output can take, and a membership function for each term. The model presented here features a single output with five fixed, non-overlapping terms that represent the transport modes among which to choose: *CAR*, *MOTORCYCLE*, *PUBLIC TRANSPORT*, *BICYCLE*, and *WALK*, as represented in Figure 6.6.

Figure 6.6: Fuzzy engine output. Each term represents a mode of transport one may choose. (Source: own research for better understanding the fuzzy logic output membership function)

Rules: Rules follow the following structure: *if INPUT is INPUT_TERM and ... then OUTPUT is OUTPUT_TERM*. The engine needs a set of rules, each with one

output but with one or more terms, as depicted in Table 6.3. Additionally, the engine provides fuzzy value modifiers or hedges through natural language keywords to help describe border cases without the need to include additional terms: *ANY*, *NOT*, *EXTREMELY*, *SELDOM*, *SOMEWHAT*, or *VERY*. The defuzzifier will be the maximum value between the three terms, that is, the term that receives the maximum value and thus the one chosen by the citizen. Please note that only *AND* conjunctions will be used between rule terms, since disjunctions can be represented by splitting a rule in two or using the *NOT* hedge.

Table 6.3: Example of fuzzy engine rule sets. (Source: own research for explaining the structure and examples of fuzzy logic rules)

if TRANSIT_DURATION is HIGH and TRANSIT_PRICE is HIGH then TRANSPORT is PRIVATE
if TRANSIT_DURATION is HIGH and PRIVATE_DURATION is LOW then TRANSPORT is PRIVATE
if PRIVATE_PRICE is HIGH and TRANSIT_PRICE is LOW and TRANSIT_DURATION is MEDIUM then TRANSPORT is TRANSIT
if WALK_LENGTH is LOW and TRANSIT_PRICE is HIGH then TRANSPORT is WALK

Expert Fuzzy Knowledge (EK) A first approach of the definition of the fuzzy logic inputs and rules was created according to both literature (Souche, 2010) and internal transport surveys carried out for the green travelling project (Staniek and Sierpiński, 2017; Sierpiński et al., 2016). From this information, a simple set of rules was extracted. These rules can be consulted in Table 6.4.

A Co-Evolutionary Fuzzy Logic Algorithm (CE) While expert knowledge can lead to accurate solutions and is crucial for a better understanding of the problem, it is not flexible enough to variations in the inputs used to build that knowledge. Therefore, it is always advisable to further search whether a better solution can be reached. Trying to manually improve the expert knowledge rules to match a wider number of citizens from the census involves applying many changes to interrelated inputs and rule sets in a very large search space. One of the approaches to automate this is to use an evolutionary algorithm and to divide the search space in subsets of problems.

Genetic programming techniques have already been used to train fuzzy logic models (Herrera, 2008). Here, a cooperative co-evolution programming method

Table 6.4: Fuzzy engine rule sets extracted from surveys and expert knowledge. (Source: own research generated from the surveys in (Staniek and Sierpiński, 2017; Sierpiński et al., 2016))

if CAR_PRICE is LOW then TRANSPORT is CAR
if CAR_TIME is LOW and TRANSIT_TIME is HIGH then TRANSPORT is CAR
if CAR_TIME is MEDIUM and TRANSIT_TIME is HIGH then TRANSPORT is CAR
if CAR_TIME is LOW and MOTORCYCLE_TIME is MEDIUM then TRANSPORT is CAR
if CAR_TIME is LOW and MOTORCYCLE_TIME is HIGH then TRANSPORT is CAR
if CAR_PRICE is LOW and TRANSIT_PRICE is LOW then TRANSPORT is CAR
if CAR_PRICE is LOW and TRANSIT_PRICE is MEDIUM then TRANSPORT is CAR
if CAR_PRICE is LOW and TRANSIT_PRICE is HIGH then TRANSPORT is CAR
if CAR_PRICE is MEDIUM and TRANSIT_PRICE is MEDIUM then TRANSPORT is CAR
if CAR_PRICE is MEDIUM and TRANSIT_PRICE is HIGH then TRANSPORT is CAR
if CAR_PRICE is HIGH and TRANSIT_PRICE is HIGH then TRANSPORT is CAR
if CAR_DISTANCE is MEDIUM and BICYCLE_DISTANCE is HIGH then TRANSPORT is CAR
if TRANSIT_PRICE is LOW and CAR_PRICE is MEDIUM then TRANSPORT is TRANSIT
if TRANSIT_PRICE is LOW and CAR_PRICE is HIGH then TRANSPORT is TRANSIT
if TRANSIT_PRICE is MEDIUM and CAR_PRICE is HIGH then TRANSPORT is TRANSIT
if TRANSIT_TIME is MEDIUM and CAR_TIME is MEDIUM then TRANSPORT is TRANSIT
if TRANSIT_TIME is MEDIUM and CAR_TIME is HIGH then TRANSPORT is TRANSIT
if TRANSIT_TIME is HIGH and CAR_TIME is HIGH then TRANSPORT is TRANSIT
if MOTORCYCLE_DISTANCE is LOW and CAR_PRICE is HIGH then TRANSPORT is MOTORCYCLE
if MOTORCYCLE_PRICE is MEDIUM and CAR_PRICE is HIGH then TRANSPORT is MOTORCYCLE
if WALK_DISTANCE is LOW then TRANSPORT is WALK
if BICYCLE_DISTANCE is LOW then TRANSPORT is BICYCLE
if BICYCLE_TIME is LOW then TRANSPORT is BICYCLE

(CE) is presented. CE is a form of evolutionary computation method that divides a large problem into sub-components and solves them independently in order to solve the large problem (Potter and De Jong, 1994). The sub-components are implemented as sub-populations and the only interaction between sub-populations is in the cooperative evaluation of each individual of the sub-populations. The cooperative evaluation of each individual in a sub-population is done by concatenating the current individual with the best individuals from the rest of the sub-populations. The algorithm presented hereby tackles both the adjustment of the input linguistic variables membership functions and the search of the rule sets that most faithfully fit the census data. These two variables are strongly related since any change in the input terms' extent requires adapting the rules to the new meaning of the linguistic

variable. Thus, the CE algorithm suits perfectly the problem of the optimising two interrelated variables in two sub-components:

Input-evolving sub-component: This is composed of a population of 200 input sets. Each input set features the four input variables already mentioned—*DURATION*, *PRICE*, *LENGTH*, and *ENVIRONMENT*. Each input has three fixed and ordered terms—*LOW*, *MEDIUM*, and *HIGH*—whose triangle anchors α, β, γ, and δ (see Figure 6.5 for details) can be modified in search of limits that better fit the citizens' appreciation for these qualitative terms.

Rule evolving sub-component: This is composed of a population of 200 rule sets. Each rule set features a list with a variable number of rules, and, for each rule, its terms and output are modified to extract the combination of rules that more faithfully represents citizens' transport choice from the census.

Each combination of input membership functions and rules defines what is called a fuzzy experiment. On evaluation, each fuzzy experiment configures the fuzzy logic engine to test how accurate the outputs of the engine are compared to the census. On the evolution of the input sets, whenever a change is applied in any of the triangle anchors, a fuzzy experiment will be created with corresponding input set, the fixed output, and the best found rule set to calculate the input set's fitness. Likewise, on the evolution of the rule sets, whenever a change is applied to the rules (whether it is an addition, a deletion, or a change in the terms), a fuzzy experiment will also be created with a corresponding rule set, and the best input set for calculating its fitness will be found.

Although evolutionary algorithm operators traditionally use mutation and crossover, rule set mutations produce only slight changes in the population. Therefore, a more abrupt approach has been designed by relying only on the crossover operator. The crossover operators for evolving rule sets and evolving input sets are detailed below.

Operators used for Rules:

rule set slice crossover Given two rule sets, this operator creates a new rule set, taking the first to n rules from the first set and the n to last rules from the second set.

rule set combine crossover Given two rule sets, this operator creates a new rule set, appending for each position a randomly selected rule from the same position of the two provided rule sets.

rule set slice terms crossover Given two rule sets, this operator clones the first rule set and selects a p rule in the set. The p rule's terms are replaced by the first to n terms from the p rule of the first set and n to last from the p rule of the second provided set.

Fuzzy Logic Input Membership Functions:

input set combine crossover Given two input sets, this function creates a new input set, appending for each position a randomly selected element from the same position of the two provided input sets.

input set shape mutation Given an input set, this function modifies one of the α, β, γ, or δ values from the shapes in Figure 6.5. It will displace at the same time the first term's triangle's third point, the second term's triangle's mid-point, and the third term's triangle's first point.

The fitness of a fuzzy experiment is given by the amount of citizens that match the transport choice originally registered in the census (that is, the accuracy of the forecasting model). The goal of the CE algorithm is, therefore, to maximize the fitness in order to estimate as many correct transport choices as possible. For this, a confusion matrix is calculated within each fitness evaluation. Confusion matrices are square matrices with a number of rows and columns equal to the number of output terms. Columns denote the amount of real elements, and rows denote the predicted elements of that class, that is, a column contains a split of 100% of the elements. Namely, cell m_{ij} denotes the percentage of elements classified as i that are of class j. The diagonal of this matrix (namely the elements m_{ii}) describes the correctly classified elements. For instance, in **??**, 19.8004% of *C* (*CAR*) instances were forecast correctly, but 14 % of *C* users were forecast as *M* (*MOTORCYCLE*) users. The global fitness is composed of the best element of the input and rules populations as seen in Figure 6.7.

Since the number of elements for each vehicle type is not balanced, a fuzzy experiment that matched only *CAR* would obtain a better fitness than other operating on a more distributed modal split. Therefore, the previously defined correctly classified elements has been improved to include balanced correctly classified elements. In this sense, let t_k denote the relative partial matches of class k, namely,

$$t_k := \frac{\sum_{i \neq k} m_{ik}}{\sum_{i=1}^{n} m_{ik}}.$$

Then, the fitness function f is defined as the sum of the logarithm of the partial matches t_k. Namely,

$$f := \sum_{k=1}^{n} \ln(1 + t_k).$$

Please note that I added 1 to the partial matches to avoid undefined values of the fitness function when a model does not correctly classify any element. Thus, with this approximation, not only are all vehicle types balanced, but rules that match the census in a more distributive way also weigh more.

Additionally, a gradually growing subset of the census is used to calculate the fitness. This subset is extended as the matched percentage of that subset increases. That is, the first iteration's fitness is calculated with a 10 % subset of the entire census data set. Following iterations, the subset will be kept equal or increased parallel to the matched percentage. In this way, as matched elements increase, more elements are introduced in the census, against which the next iteration's fitness can be evaluated. Moreover, in order to go through all the elements of the census, at every 15 iterations, the subset to extract from the census is moved by 10 %. This changes in the reference data with which to evaluate the fitness and generates the repeating pattern visible in Figure 6.7.

As a summary, the pseudo code for the CE algorithm is described in **??**. The algorithm is divided into two main procedures:

1. initialise the population of rule sets and input sets that will take part in the co-evolutionary process, and

2. initialise the co-evolutionary process itself where the rule sets and input sets evolve together through the application of the evolutionary operators, the calculation of the fuzzy experiments' fitness on each iteration, and the update of the percentage of census being used for evaluation.

Random Search (RA) In order to assess whether the results had been produced by chance or, indeed, due to the evolutionary process, a simple yet powerful method, random search, was also trained and validated. For this purpose, the same amount of random fuzzy models as that of fitness evaluations for the CE were produced. As 200 rules are in the population and 200 generations have been made, 400,000 random search rules have been generated. All models, in general, and the CE algorithm, in particular, are expected to produce far better results than this method.

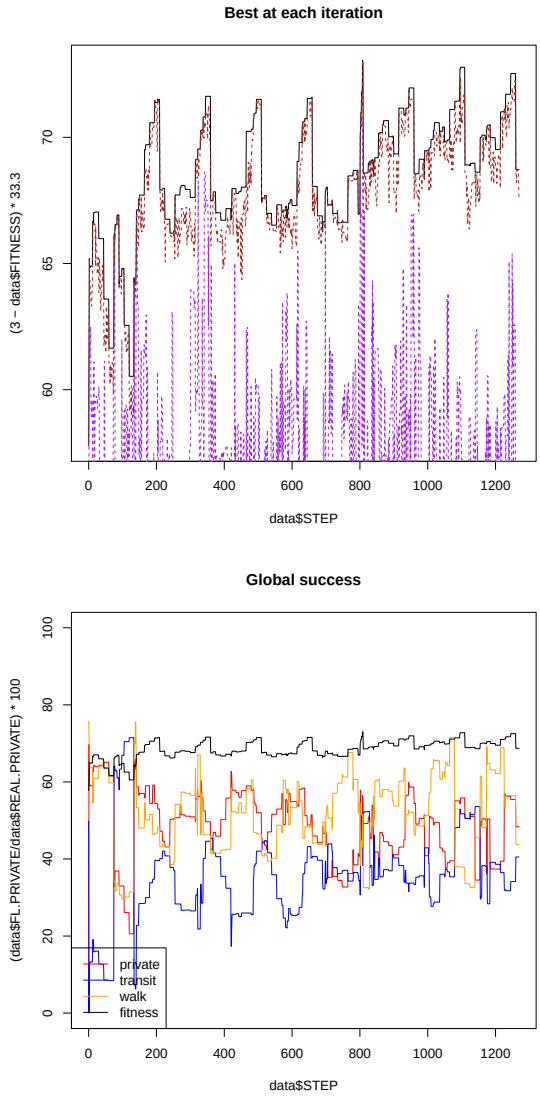


Figure 6.7: Iterations of the co-evolutive fuzzy algorithm. (Source: own research by plotting the evolution in time of the algorithm)

6.2 Modelling physical reality

In order to prepare an scenario as similar as possible to the real world where active citizens must decide which transport to use, the simulation comprises the identification, definition, connection and parameterisation of large amount of agents. In this simulation the following passive entities have been modelled:

Administrative area: Geographical location of municipalities, neighbourhoods, or administrative boundaries on which departure and arrival points for daily commutes are defined.

Building: Buildings within the administrative areas for citizens to be distributed are proportionately in accordance to the area and building levels.

Transport Lines: Network that enables the movement of agents through the environment. A set of transport lines of the same type make up a specific transport graph. GeoWorldSim creates several transport graphs according to the different type of vehicles available in the simulation scenario (depicted in Figure 6.8). Therefore, a graph is created for roads, another for railways and, finally, another graph for pedestrians. Agents are only able to travel through one of these graphs, depending on their type. Additionally, each TransportLine has a `maximum speed, length, allowed agent types, crossing price` and the skill to host one or more vehicles inside according to its length and lanes.

Public Transport Stops: Represent special connection nodes shared by two graphs, making it possible for people to change between means of transport. If the stop is a Bus Stop, it will be connected to the Roads and Walk graphs whereas if it is a Rail, Tram, Subway or Funicular stop, it will be connected to the Railway and Walk graphs. It is important to note that people walking to public transport stops cannot continue their journey by walking on the rail or roads. Stops have the skill to contain waiting people and emit signals whenever a public transport vehicle arrives or departs. These notifications contain information about the vehicle arriving, its driver, and a list of remaining stops. Therefore, all users that want to use a specific public transport, can enter the Public Transport

Stop, subscribe to its notifications, and wait for the train, bus, or tram, to arrive. When the users receive a notification of vehicle arrival, they look for their destination in the list of remaining stops, and decide whether to get on the public transport vehicle or not.

Traffic Signals: These are connected to their nearest road graph node. They alternately close and open all entering Roads and Walk Paths every 15 seconds, so as to manage the crossing permissions of vehicles and pedestrians.

Figure 6.8: Transport network according to their type: Roads, Railways and Footways. (Source: own research)

Conversely in the simulation the active agents have been:

Private Transport Vehicles: Private vehicles (cars, electric cars, motorcycles, and bicycles) that people can use to travel through the scenario. Vehicles have two important skills: a move skill that allows them to change position along a road, and a produce pollution skill that generates contaminant emissions and consumes fuel according to the emissions model described in (Pijoan et al., 2017b).

Public Transport Vehicles: An extension of normal vehicles, the public transport vehicles emit signals to notify their occupants of the arrival to a public transport stop. In this way, the passengers inside the vehicle will connect to know whether they have arrived at their destination.

Public Transport Drivers: People that drive the public transport vehicles along the transport lines. For public transport itineraries definition, drivers are created from files supplied by the transport operators in General Transit Feed Specification (GTFS) (Antrim et al., 2013) format. These drivers will follow the theoretical

timetable scheduled in the GTFS file, stopping at each public transport stop to pick up or drop passengers.

Citizens: Agents that emulate persons in the daily travelling between their houses and their work or study places. Section 6.1 has already covered the main description for their behaviour.

All these represent the minimum necessary baseline from which to emulate the real world. The main characters of the Geosimulation, however, are the citizens. Each person in the simulation will have a different profile based on their personal preferences which will directly influence the transport mean selected for travelling. For the experimentation, census data has been used in order to build these profiles.

A profile is composed of an origin point, a destination point, a departure time, and a set of all the transport means available to the citizen. Prior to each simulation, the platform preprocesses the input data to geolocate each Citizen agent on the environment by performing a weighted distribution. This process assigns the origin of each Citizen agent to buildings in the municipality and the destination to available amenities, commercial or industrial buildings, based on the building size and levels, i.e. the bigger and taller the building, the bigger the probability of assigning a citizen to it. Once located, Citizen agents are scheduled to wake up at an specific time and initiate their daily commutes by selecting the appropriate transports that meet their needs.

All Citizen agents are executed together to emerge everyday traffic congestion, traffic light stopping times and how these affect time travels and emissions. The Geosimulation also makes possible to compare the estimated travel times with the real ones resultant from bringing together all the commutes.

6.2.1 **Multi-modal router and incentives**

Getting to make the right decision involves having precise data of all available transport options each citizen has. Citizens need to calculate the travelling options they have available and are willing to use, for which their itineraries will be calculated. All the itineraries a citizen can perform from its departure to arrival points have been

calculated using GTPlanner. This is a multi-modal algorithm (Sierpiński et al., 2016) that is capable of providing routes which combine transfers between vehicles and transport modes. GTPlanner calculates itineraries for *CAR*, *MOTORCYCLE*, *PUBLIC TRANSPORT*, *BICYCLE*, *WALK*, *ELECTRIC CAR*, *PARK & RIDE*, *BIKE & RIDE*, *URBAN CAR* and *URBAN BICYCLE*. It returns itineraries divided into legs according to the type of vehicles being combined. For each leg, the algorithm sets a starting and an ending point, a transport mean, journey duration, journey length, journey price, and estimated pollutant emissions. A simplified public transport itinerary response contains the information shown in Listing 6.13.

While Citizen agents calculate their itinerary legs, the objective of the Geosimulation is to quantify how the application of different incentives or restrictions over these legs affect their final choice. It is important to take into consideration that the current research focuses on the evaluation of non-physical incentives, such as changes in the price of the journey, whether it is a discount for using public transport or an additional tax for using private vehicles, reductions in the duration of the journey, or the application of congestion charges. Even if GeoWorldSim allows to assess physical changes like modifying public transport lines or modifying infrastructures, there is no option for applying on-the-fly network changes to the multi-modal algorithm.

Table 6.5 describes some of the objectives (Council, 2012) different City Town Halls in Biscay plan to apply in order to foster the use of greener means of transport and avoid congestion inside the city. Incentives are loaded into GeoWorldSim using a set of keywords that modify the features of the itinerary legs calculated by the multi-modal algorithm. For example, reducing public transport price would alter only the legs concerning public transport or having to pay a 5âĆň congestion charge to enter the city, would increase private car leg costs. Once all the incentives have been applied to each itinerary leg, it is time for the transport selection model to chose the transportation for each citizen according to their preferences.

Incentive/Restriction	Feature altered
Reduce public transport price by half	$put\_price := put\_price * 0.5$
Make all public transport free	$put\_price := 0$
Reduce transfer waiting time by half	$transfer\_time := transfer\_time * 0.5$
Free parking for electric vehicles	$electric\_parking := 0$
A 5€ congestion charge to vehicles entering the city	$car\_price := car\_price + 5$

Table 6.5: Incentives and what attributes they modify

```
"itinerary":
   {
    "duration" : 4211, "startTime" : 1490257800000,
    "endTime" : 1490260211000, "walkTime" : 1092,
    "transitTime" : 1311, "waitingTime" : 8,
    "walkDistance" : 1273.6906992374816, "transfers" : 1,
    "pollution_CC" : 0.003567395424801314,
    "legs":[
      {
       "mode" : "WALK", "start" : 1490257800000,
       "end" : 1490258546000, "distance" : 911.814,
       "from" : { "lon" : -3.0082774176765663, "lat" : 43.30321750209903 },
       "to" : { "lon" : -3.0137010338329, "lat" : 43.2989136508072 }
      },
      {
       "mode" : "BUS", "start" : 1490258547000,
       "end" : 1490258827000, "distance" : 4823.12,
       "from" : { "lon" : -3.0137010338329, "lat" : 43.2989136508072 },
       "to" : { "lon" : -2.98918424176295, "lat" : 43.2932903929508 }
      },
      {
       "mode" : "WALK", "start" : 1490258828000,
       "end" : 1490258968000, "distance" : 294.23,
       "from" : { "lon" : -2.98918424176295, "lat" : 43.2932903929508 },
       "to" : { "lon" : -2.98936, "lat" : 43.2956 }
      },
      {
       "mode" : "SUBWAY", "start" : 1490258973000,
       "end" : 1490260004000, "distance" : 2173.905,
       "from" : { "lon" : -2.98936, "lat":43.2956 },
       "to" : { "lon" : -2.92071, "lat" : 43.2597 }
      },
      {
       "mode" : "WALK", "start" : 1490260005000,
       "end" : 1490260211000, "distance" : 305.04,
       "from" : { "lon" : -2.92071, "lat" : 43.2597 },
       "to" : { "lat" : 43.25731364536219, "arrival" : 1490260211000 },
      }
    ]
}
```

Code 6.13: Response from the multi-modal GTPlanner algorithm.

6.3 Results

This section presents the results for comparing the eight data science methods described in Section 6.1.1 and afterwards executing the Geosimulation using the Fuzzy Logic model. The experiments were carried out using the following:

- The GeoWorldSim platform (Pijoan et al., 2015b) with the open-source fuzzy logic library Fuzzylite (Rada-Vilela, 2018).
- The Caret package (Kuhn, 2017) of the R Statistical Suite (R Core Team, 2017). This package uniforms the access to several libraries and stages needed to build data-based models. For example, it provides a uniform way to tune the hyper-parameters, train the models, and produce a forecast among other facilities.

Simulations were executed on two computers: large experiments were run on a system with an AMD Opteron 6168 CPU and 32 GB of RAM, and short experiments were run on a system with an Intel Core i7-2600 CPU and 16 GB of RAM.

Table 6.6 shows the global accuracy results, that is, the number of trips correctly classified. The first row *(5-TM)* shows the results with the five transport modes, while the second row *(3-TM)* shows the results after grouping *CAR* and *MOTORCYCLE* into *PRIVATE VEHICLE* as well as *WALK* and *BICYCLE* into *WALK*.

Table 6.6: Global accuracy (%) of the algorithm. Column T (testing) shows the results for Biscay's data set and Column V (validation) for Silesia. (Source: own research by extracting the forecast accuracy of the different models in Table 6.8. EK: expert knowledge; CE: fuzzy logic; SVM: support vector machine; M: multinomial logit; NN: neural network; B: naive Bayes; KNN: knearest neighbor; RA: random search.)

	EK		**CE**		**SVM**		**M**		**NN**		**B**		**KNN**		**RA**	
	T	***V***	***T***	***V***	***T***	***V***	***T***	***V***	***T***	***V***	***T***	***V***	***T***	***V***	***T***	***V***
5-TM	24	22	37	40	51	50	50	49	50	48	36	28	63	47	14	2.7
3-TM	47	45	49	46	51	49	51	51	51	46	38	34	64	48	31	26

As seen, all techniques are able to transfer the results almost seamlessly since the differences between the accuracy of the training (T) and validation (V) sets are

rather small. As explained previously, each method was *trained* with data from the census of Biscay (Basque Country, Spain) and *validated* over the data set of Silesian Voivodeship (Poland). The models can be classified in three groups: the cluster of the best models, the contrast models, and the rest of the models. As expected, the worst model is the random search (RA). The best models are the most complex ones: SVM, NN, and M. Surprisingly, the KNN method, while being very simple, achieves the best results. The rest of the models—EK, CE, and B—complete the clusters of the other models. In all cases, the best models can correctly predict the transport choice of almost half of the trips registered in the data set.

For more detail, Table 6.8 contains the confusion matrices of all models included in this article. As previously stated, within these matrices, columns show the observed modes, and rows show the forecast real modes. In theses tables, it is clear that some of the models have difficulties differentiating between *CAR* - *MOTORCYCLE* and *WALK* - *BICYCLE*. This may be caused by the existence of other additional variables, such as socio-demographics, car/motorcycle ownership, or climatology factors that are not taken into consideration in this article. Therefore, since the census lacked information about these features, the five transport modes were grouped into *three categories*. This way, *CAR* and *MOTORCYCLE* were gathered into *PRIVATE VEHICLE*, *WALK* - *BICYCLE* into *WALK*, and *TRANSIT* is maintained as its own class.

With these three transport modes, all techniques are able to correctly transfer the results from the training to the validation set (Table 6.6). As before, three clusters of techniques appear. On the one hand, RA continues to be the worst method, but it is close to B. On the other hand, EK has greatly improved its performance and produces the best cluster. In any case, the best models have not improved their performance, yet they are still able to correctly predict the transport choice of half of the trips made.

In order to validate the previous claims, a bootstrapping validation was used (Efron, 1979). Thus, 100 samples were randomly built following the original distribution for the Silesia data sets. Figure 6.9 shows a box plot with the results of the different models bringing to light the significant differences among them. In fact, a Friedman test confirms this hypothesis (p-value $< 10^{-16}$). In order to assess the differences among the different models, a post-hoc analysis has been made following the procedure

described in (Piepho, 2004). This procedure clusters all methods with similar results and gives them the same letter.

It should be taken into consideration that these groups are not sorted, namely, the best algorithms are not labeled with an *a*. The results are shown in Figure 6.10. In this figure, the mean value of the different algorithms is plotted and groups are identified with a letter. As can be seen, the test confirm our claims. The three groups are clearly visible.

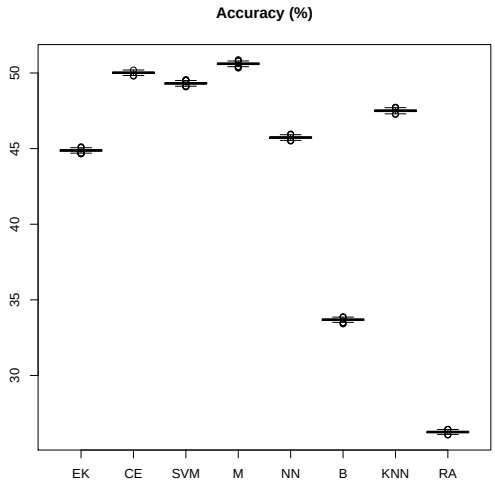


Figure 6.9: Box plot of the distribution of the models' accuracy of the three transport modes on 100 bootstrapping samples. In Biscay, the probability of using *BICYCLE*, *CAR*, *MOTORCYCLE*, *TRANSIT*, and *WALK* is 1.75%, 43.51%, 2.1%, 37.6%, and 15.04%, respectively. In Silesia, the probability of using *BICYCLE*, *CAR*, *MOTORCYCLE*, *TRANSIT*, and *WALK* is 0.82%, 53.96%, 0.12%, 36.33%, and 8.75%, respectively. Therefore, if a model can produce such a forecast, it will achieve an accuracy of 100 %. (Source: own research generated from R Statistical Suite (R Core Team, 2017))

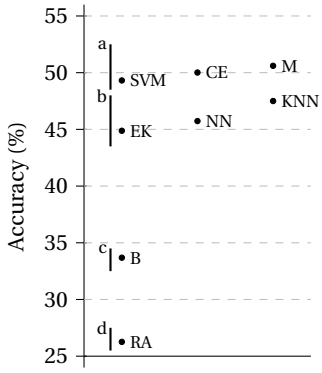


Figure 6.10: Results of an ALL vs. ALL post-hoc analysis of the accuracy of the models. (Source: own research by plotting the results from R Statistical Suite (R Core Team, 2017))

Again, when analysing the results in detail, problems arise for some of the models. Looking at the confusion matrices (Table 6.9) highlights the fact that some models continue to have problems with some of the classes. For example, the SVM cannot correctly predict any *WALK* trip (even if it has correctly predicted half of the trips). The same problem occurs, when looking at the M and NN models, though it is not as acute. In the end, it seems that only the KNN, EK, and CE models do not suffer these problems in the setting.

Finally, Table 6.7 shows the forecasts that the different models share. As before, 100 bootstrapping samples were built from the Silesia data set. Column Real shows the results of the survey carried out in 2015 (Green Travelling Consortium, 2013), and Columns L.C.I. and U.C.I. contain the lower and upper confidence interval at the 0.050 significant level ($\alpha = 0.5$) for the forecast value made. As expected, given the low variance of the bootstrapped samples (see Figure 6.9) the modal split forecasts of all models are quite similar (the confidence intervals are quite narrow). Nevertheless, the real value is not included in the confidence interval.

Model	Transport Mean	Real	L.C.I	U.C.I.
EK	WALK	9.57	31.10	31.12
EK	TRANSIT	36.33	0.08	0.08
EK	PRIVATE	54.10	68.80	68.85
CE	WALK	9.57	6.89	6.90
CE	TRANSIT	36.33	19.20	19.23
CE	PRIVATE	54.10	73.87	73.91
SVM	WALK	9.57	0.00	0.00
SVM	TRANSIT	36.33	41.15	41.17
SVM	PRIVATE	54.10	58.81	58.85
M	WALK	9.57	0.11	0.11
M	TRANSIT	36.33	33.11	33.16
M	PRIVATE	54.10	66.75	66.77
NN	WALK	9.57	4.36	4.38
NN	TRANSIT	36.33	49.63	49.65
NN	PRIVATE	54.10	45.99	46.01
B	WALK	9.57	46.14	46.16
B	TRANSIT	36.33	21.79	21.81
B	PRIVATE	54.10	32.05	32.08
KNN	WALK	9.57	7.47	7.48
KNN	TRANSIT	36.33	35.61	35.65
KNN	PRIVATE	54.10	56.88	56.92
RA	WALK	9.57	44.46	44.48
RA	TRANSIT	36.33	35.57	35.59
RA	PRIVATE	54.10	19.94	19.96

Table 6.7: Bootstraping values (in %) for the modal split forecast by the different models. (Source: own research)

6.3.1 **Geosimulation results**

Once modelled the transport choice of agents, this subsection presents the results obtained by Geosimulating the base scenario of Biscay using the expert fuzzy logic rules and how much it behaves as the real citizens. The Geosimulation yields the following results in terms of modal split Figure 6.11 and travel times Figure 6.12.

Comparing the two modal splits, the amount of citizens that choose *CAR* is somewhat higher than that of the real census. In the same way the amount that choose *WALK* is higher, contrasting heavily with the amount of citizens that choose *MOTORCYCLE* or *BICYCLE*. I believe that the transport choice model is pushing private vehicle users to *CAR* instead of *MOTORCYCLE* due to the fact that the values for distance and time are similar. The same concept with distances and prices can be applied to people being pushed from *BICYCLE* to *WALK*.

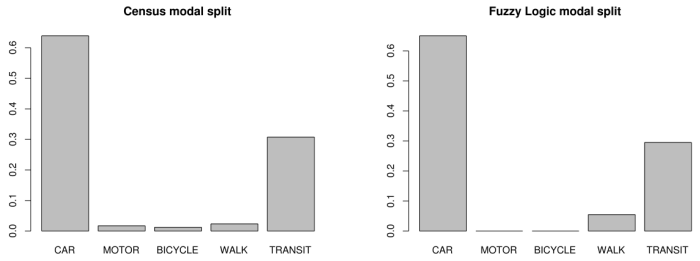


Figure 6.11: Census real modal split and expert fuzzy logic obtained modal split. (Source: Spanish National Statistics Institute (Instituto Nacional de Estadística, 2011) and own research)

Regarding travel time and duration, launching the Geosimulation achieves a scenario where all agents run in parallel generating traffic jams with more reliable results. Figure 6.12 outlines that private vehicle is the less time consuming option for home-to-work commutes. However, these do not maintain the best average speed, since that position belongs to *SUBWAY* which does not lose time in traffic jams, crossings and signals. Thanks to traffic jams, *MOTORCYCLE* achieves a slightly higher average speed as they can overtake stopped cars.

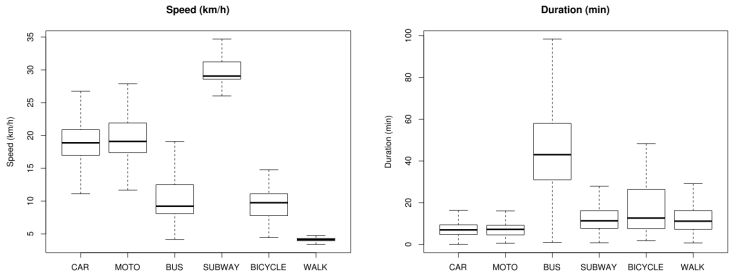


Figure 6.12: Obtained commutes speed and duration per vehicle type (Source: own research)

Figure 6.13 highlights the hegemony of private vehicle both for inter and extra municipal displacements. All private vehicle commutes (*CAR* and *MOTORCYCLE*) can be made in less than 40 minutes, a result consistent with the distance to the municipalities evaluated. It can be seen that quite a few people also choose the private car as a means of transportation for inter-municipal trips.

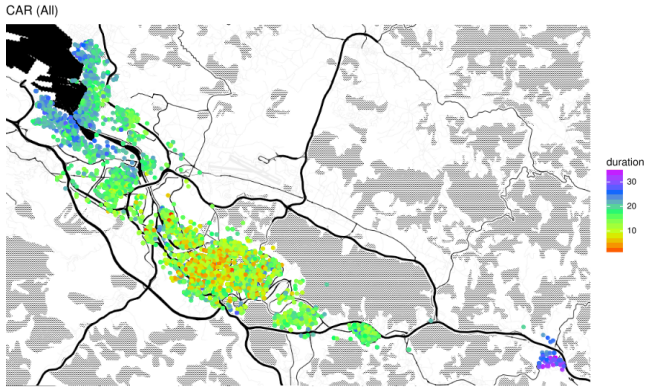


Figure 6.13: Obtained private car commutes geolocated (Source: own research)

Figure 6.14 that there is a large part of the population of the census that chooses public transport for inter-municipal and extra-municipal commutes. Unlike with private vehicle, journey times take much longer according to the distance to public transport stops and frequency of the service. This last one is the factor why individuals who depart from similar locations take a very different time in completing the commutes as they have to wait for their specific *BUS* or *SUBWAY* to arrive.

Finally, Figure 6.15 shows that walking option has only been chosen for mainly inter-municipal commutes that is for short commutes. However some walking times exceed one-and-a-half hours which suggests having to improve the fuzzy logic transport choice model.

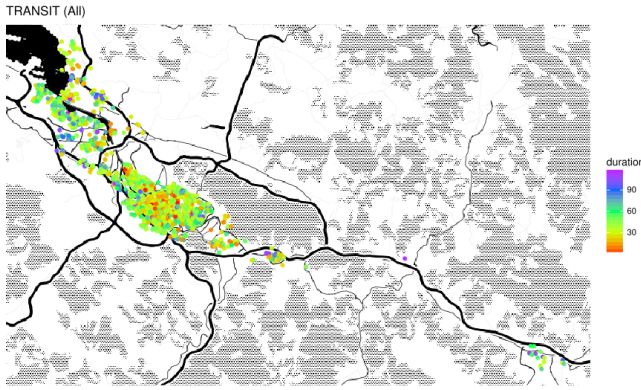


Figure 6.14: Obtained public transport commutes geolocated (Source: own research)

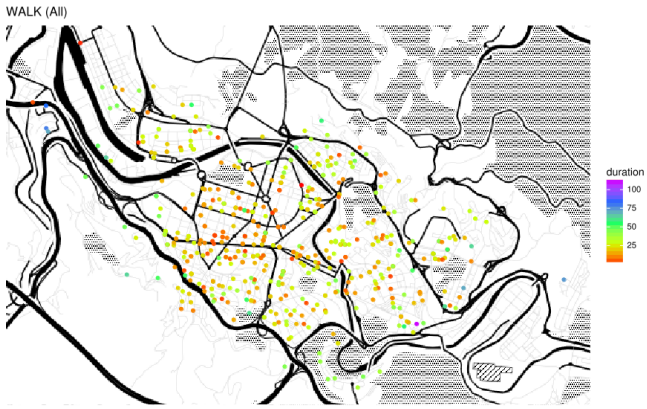


Figure 6.15: Obtained walking commutes geolocated (Source: own research)

6.4 Conclusions and Future work

These results, though at an early stage, suggest that the methodology here proposed is promising for evaluating the impact of applying green travelling policies. As explained in Section 6.3, the results suggest the existence of three groups of models: On the one hand, as expected, the most complex models tested (SVM, CE, and M) perform quite well, hitting almost half of the trips. Next, there is another set of complex algorithms but with slightly worse results (NN, EK, and KNN). The results of the KNN algorithm was in fact unexpected, as it is a very simple algorithm. On the other hand, the control methods (B and RA) do not achieve good results, as expected.

In order to draw meaningful conclusions, a qualitative analysis should also be made:

- Some of the models produce unbalanced predictions, like M, SVM and NN, which completely ignore some of the transport modes. Therefore, given the similar prediction capabilities, the models with more balanced predictions should be used, namely, the EK, CE, and KNN models.
- The number of parameters and complexity of the models are not the same either. In this sense, simpler models should be preferred to more complex models. However, EK, CE, and KNN are non-parametric, and their complexity is difficult to assess. On the one hand, the number of parameters of KNN is indeed the amount of points in the training data set, but the complexity of the model is quite low (understanding here the complexity in terms of its VC dimension or similar measures (Vapnik and Chervonenkis, 1971)). On the other hand, the amount of parameters in the EK and CE models is quite low (compared to the number of parameters of KNN), but the complexity of the model is higher (Chen and Wang, 2003).
- Comparing the EK and CE fuzzy rule sets, both have similar amounts of rules (EK has 33 and CE has 30). However, the EK rules are only composed of one or two terms, which makes it easy to follow and modify, while the CE evolved rules usually have about 10 or more terms with a relation between them that is not very clear.

- It is always better to use a model that is easy to be understood than a data-driven model. Following this advice, EK and M should be preferred to the rest of the models.

Based on the above considerations, it is clear that the best model is EK, as it is possible to be understood, is easy to be extended so as to cover new transport policies, has a complexity that is similar to other models that produce similar numerical results, and its predictions are well balanced among the classes.

Even though the results are good, the models need to be improved. At the moment, the methodology does not take into account socio-economic variables, which are not present on the tested data sets (see Table 6.2). Recent works ((Khattak et al., 2017; Ding and Zhang, 2016)) have achieved successful results by using demographic and socio-economic features. As shown in (Ding and Zhang, 2016) (where they have almost reached a 90 % chance of success), socio-economic information from commuters enables the clustering and creation of custom utility functions for each group. Furthermore, climatic variables have been identified as relevant (Chiu Chuen et al., 2014). Future refinements of the model may introduce both sets of variables in search of reducing error. A first approach should consist of adding these non-trip related features to the global census models and evaluate their improvement. In case this is not satisfactory, the next step will be to try clustering the commuters and adjusting the models for each group.

Additionally, the ownership of private vehicles is a relevant parameter not detailed in the used data sets. In order to introduce this information, the model will use Monte Carlo simulations so that several distributions of these variables are simulated and the results assessed.

Finally, once the transport choice model is fitted, the next step is to determine the extent to which different non-physical incentives, such as changes in the price of the journey, discounts on public transport, additional taxes on the use of private vehicles, reductions in the duration of the journey, and congestion charges, affect commuters' choices. These incentives alter the features of the itineraries, resulting in a different modal split. In order to check the suitability of these incentives, the citizen agents within the simulation will be given a set of preferences that will determine whether

they are prone to accept or ignore a certain policy, based on the itinerary features already modelled in the transport choice model. This approach will involve defining the thresholds at which the applications of transport policies will be effective, all the while seeking an equilibrium between the objectives of the policies and the citizens' preferences in order to avoid social rejection or oversizing.

6.5 **Result tables**

Table 6.8: Confusion matrices for all the models for the five transport modes (in %). (Source: own research)

(**a**) Expert Knowledge Confusion Matrix

		Biscay Observed					Silesia Observed				
		B	**C**	**M**	**T**	**W**	**B**	**C**	**M**	**T**	**W**
Forecast	**B**	21	26	14	17	23	40	6.3	0	0.000	50
	C	14	20	12	19	6.9	3.6	24	8.5	5.1	54
	M	19	14	24	21	23	13	8.2	21	1.7	12
	T	13	18	15	14	4.2	16	12	60	24	34
	W	33	22	36	29	43	27	49	10	69	0.69

(**b**) Co-evolutionary Fuzzy Algorithm Confusion Matrix

		Biscay Observed					Silesia Observed				
		B	**C**	**M**	**T**	**W**	**B**	**C**	**M**	**T**	**W**
Forecast	**B**	26	18	29	23	21	4.3	3.7	3.2	4.1	5.2
	C	5.0	26	3.9	5.0	3.7	43	56	21	50	14
	M	0	0	0	0	0	46	36	65	40	48
	T	24	31	25	29	5.7	0.39	0.14	0	0.14	0.55
	W	45	24	43	43	70	5.7	4.9	11	5.4	32

(**c**) Support Vector Machine Confusion Matrix

		Biscay Observed					Silesia Observed				
		B	**C**	**M**	**T**	**W**	**B**	**C**	**M**	**T**	**W**
Forecast	**B**	0	0	0	0	0	0	0	0	0	0
	C	34	64	37	38	11	62	69	44	66	77
	M	0	0	0	0	0	0	0	0	0	0
	T	66	36	63	62	89	38	31	56	34	23
	W	0	0	0	0	0	0	0	0	0	0

(**d**) Multinomial Confusion Matrix

		Biscay Observed					Silesia Observed				
		B	**C**	**M**	**T**	**W**	**B**	**C**	**M**	**T**	**W**
Forecast	**B**	3.0	0	0	0	0	0	0	0	0	0
	C	34	65	120	2300	250	63	71	44	68	16
	M	0	0	0	0	0	0.18	0.65	0	0.53	0.0051
	T	63	35	190	3200	2000	37	28	56	31	84
	W	0	0.078	0	1	0	0.11	0.074	0	0.072	0.44

(**e**) Neural Network Confusion Matrix

		Biscay Observed					Silesia Observed				
		B	**C**	**M**	**T**	**W**	**B**	**C**	**M**	**T**	**W**
Forecast	**B**	0	0.016	0	0	0	0.026	0.023	0	0.032	0.049
	C	33	64	120	2300	300	43	52	26	49	17
	M	0	0	0	0	0	0.11	0.062	0	0.059	0.25
	T	62	36	190	3300	1900	57	47	73	51	81
	W	1.5	0.34	2	36	36	0.37	0.45	0.98	0.51	1.1

(**f**) Naïve Bayesian Confusion Matrix

		Biscay Observed					Silesia Observed				
		B	**C**	**M**	**T**	**W**	**B**	**C**	**M**	**T**	**W**
Forecast	**B**	3.4	0.28	0.32	0.22	0	0.92	2.4	1.6	1.1	0.85
	C	9.4	30	7.7	7.5	4.2	27	32	18	31	18
	M	0	0	0	0	0	0.76	0.56	0	0.60	0.70
	T	19	27	20	26	5.1	19	24	7.8	22	0.12
	W	68	43	72	66	91	53	41	73	46	81

(**g**) *k*-Nearest Neighbours Confusion Matrix

		Biscay Observed					Silesia Observed				
		B	**C**	**M**	**T**	**W**	**B**	**C**	**M**	**T**	**W**
Forecast	**B**	0	0	0	0	0	0	0	0	0	0
	C	33	64	37	33	21	53	62	38	59	27
	M	0	0	0	0	0	0	0	0	0	0
	T	54	32	55	61	61	45	36	59	39	62
	W	12	3.9	8.1	6.1	18	2.3	2.1	2.9	2.3	12

(**h**) Random Search Confusion Matrix

		Biscay Observed					Silesia Observed				
		B	**C**	**M**	**T**	**W**	**B**	**C**	**M**	**T**	**W**
Forecast	**B**	0	0	0	0	0	0	0	0	0	0
	C	100	100	100	100	100	0	0.14	2.0	5.2	23
	M	0	0	0	0	0	100	86	98	91	76
	T	0	0	0	0	0	0	0	0	3.7	0
	W	0	0	0	0	0	0	14	0	0	0.77

Table 6.9: Confusion matrices for all the models for the three transport modes (in %). (Source: own research)

(**a**) Expert Knowledge Confusion Matrix

		Biscay Observed			Silesia Observed		
		P	**T**	**W**	**P**	**T**	**W**
Forecast	**P**	27	8.6	11	64	60	33
	T	48	52	28	0.053	0.059	0.47
	W	25	40	62	36	39	67

(**b**) Co-evolutionary Confusion Matrix

		Biscay Observed			Silesia Observed		
		P	**T**	**W**	**P**	**T**	**W**
Forecast	**P**	28	7.0	5.3	23	26	15
	T	45	46	21	41	55	60
	W	27	47	74	36	18	25

(**c**) Suport Vector Machine Confusion Matrix

		Biscay Observed			Silesia Observed		
		P	**T**	**W**	**P**	**T**	**W**
Forecast	**P**	64	40	14	64	60	20
	T	36	60	86	36	40	80
	W	0	0	0.12	0	0	0

(**d**) Multinomial Confusion Matrix

		Biscay Observed			Silesia Observed		
		P	**T**	**W**	**P**	**T**	**W**
Forecast	**P**	66	44	15	77	74	23
	T	34	56	85	23	26	77
	W	0.074	0.018	0	0.12	0.13	0.48

(**e**) Neural Network Confusion Matrix

		Biscay Observed			Silesia Observed		
		P	**T**	**W**	**P**	**T**	**W**
Forecast	**P**	58	36	15	47	47	25
	T	42	63	84	52	52	71
	W	0.39	0.41	1.1	1.1	1.0	4.3

(**f**) Naïve Bayes Confusion Matrix

		Biscay Observed			Silesia Observed		
		P	**T**	**W**	**P**	**T**	**W**
Forecast	**P**	29	7.7	4.8	35	32	20
	T	27	26	6.6	24	22	1.8
	W	44	66	89	41	46	78

(**g**) Random Search Confusion Matrix

		Biscay Observed			Silesia Observed		
		P	**T**	**W**	**P**	**T**	**W**
Forecast	**P**	100	100	100	22	100	100
	T	0	0	0	78	0	0
	W	0.063	0.057	0	0	0	0

(**h**) K-Nearest Neighbours Confusion Matrix

		Biscay Observed			Silesia Observed		
		P	**T**	**W**	**P**	**T**	**W**
Forecast	**P**	66	36	25	64	61	32
	T	29	56	53	33	36	55
	W	4.9	7.5	21	2.8	3.0	13

CHAPTER

7

Use-case 3 - Long term Load forecasting

Electrical distribution companies, face a big challenge when trying to predict and evaluate the growth in demand for electrical power. Under the current volatile economic conditions, this problematic boosts exponentially: distribution companies aim at getting the most out of the existing infrastructures, especially when their renovation can be really expensive. Thus, long term load forecasts are extremely useful for energy suppliers, Distribution System Operator (DSO), financial institutions, and other participants in the electric energy generation, distribution, and retail sectors.

The concept of Long Term Load Forecasting (LTLF) involves social, economic, policy and technical issues to which we must add the problems of having limited amounts of information and the the difficulty to operate with the scarce data available (Willis, 1996). In this sense, LTLF is closely linked to urban evolution since its main models derive from the fields of geography and sociology. Nowadays, these models are being combined with others stemming from Artificial Intelligence (AI).

This integration brings together the best qualities of both areas: AI provides the learning ability and the capacity to interpret the available data, while social and spatial models define the natural behaviour and characteristics of the problem at hand. The technical literature shows a wide range of methodologies and models for LTLF which can generally be classified into two broad categories: statistical methods and artificial-intelligence-based methods. Statistical load forecasting methods comprise approaches mainly based on time series and regression models (Alfares and Nazeeruddin, 2002). In turn, artificial intelligence methods comprise methodologies like networks (Senjyu et al., 2002), genetic algorithms (Lai, 1998), support vector machines (Pai and Hong, 2005), and fuzzy logic (Farahat, 2004).

The growth in demand for electric power can be decomposed into two axes. On the one hand, there is the so-called *vertical growth*, which represents a boost in power demand due to an increment in the electrification of the existing households. On the other hand, the *horizontal growth* involves the appearance of new settlements on an specific area, motivated by the natural evolution of the population. The analysis of these phenomena is crucial since an improper estimation may lead to the saturation of the electrical infrastructure and the loss of power supply, along with the consequent economic losses and social distress.

In this venue, there is a special type of LTLF that deserves a closer look due to its economic importance: Spatial Load Forecasting (SLF). Usually, SLF uses a model built over a GIS to get together data related to electric distribution, land use, and development indicators. In this way, urban infrastructure engineers will be able to predict, years in advance, any new load and how it affects the electric system, helping them to determine whether the current infrastructure should be upgraded or not. Failing to do so leads to the inability to cope with load peaks, appearance of brownouts, blackouts and, in general, low-quality supply. Furthermore, there is a lengthy amount of previous research on agent-based simulations using GIS, whether it is a geographic phenomena or a phenomena with an important geographic component, covering several aspects of Urban Modelling (Li and Muller, 2007; Crooks et al., 2008b) and Housing (Jordan et al., 2010). Agent-Based Modelling (ABM) has

also experimented a notable boost in the SLF field due to its capacity to model the future evolution of energy demand on a certain zone.

The Geosimulation built on GeoWorldSim presented in this chapter is able to simulate the variation in the load of the transformers and electrical substations located on a certain city. To this end,we have modelled the behaviour, evaluations and decisions a human takes when choosing a new place to live. The configuration is based on the environmental approach proposed by *Russel and Norvig* (Russell et al., 2003; Weyns et al., 2004) being:

Accessible: The agents have access to the whole environment.

Non-deterministic: A change in the state of the environment depends on the management of threads by the Operating System on which the simulation is running.

Dynamic: The environment can change while the agent deliberated.

Discrete: The number of percepts is limited and centralised.

The Environment is populated by both Passive Entities, used for representing the city infrastructure like substations, transformers and existing buildings, and Autonomous Agents in the form of greenfields and buyers. In line with this model, the pseudo code can be described as follows:

```
Data: Initial state of the environment
while !termination(state) do
    for greenfield in greenfields do
        PERCEPT1[greenfield] = Get-Around-Facilities(greenfield, state)
        PERCEPT2[greenfield] = Get-Around-Constructed-Buildings(greenfield,
          state)
        ACTION1[greenfield] =
          Determine-Dwelling-Characteristics[PERCEPT1,PERCEPT2(agent)]
        ACTION2[greenfield] = Set-Dwelling-Price[ACTION1(agent)]
    end
    for agent in agents do
        PERCEPT[agent] = Get-List-Greenfields(agent, state)
        ACTION1[agent] = Evaluate-Greenfields[PERCEPT(agent)]
        while !assigned && greenfields-available do
            ACTION2[agent] = Get-Greenfield[ACTION1(agent)]
        end
        ACTION3[CT] = Add-Load[ACTION2(agent)]
    end
    state = UPDATE-FN(actions, agent, state)
end
Procedure RUN-ENVIRONMENT(state, UPDATE-FN, agents, termination)
```

7.1 Modelling social behaviour

The buyers are agents that emulate the people looking for a new house. Since every person has different preferences about the presence of (or distance to) a particular public facility, these have been encoded in a vector a_i that describes how important each infrastructure is to a particular agent i. Moreover, agents have an individual budget limit and a degree of greediness depending on which, they will query a different number of greenfield. Further, we have identified three primary target groups sharing a common preference pattern: Elderlies, Families and Singles. The accurate values of the preference vector have been issued using a uniform random variable with the mean described in Table 7.1 and a 10% of standard deviation.

When the agents have been loaded into the Environment, they select a number of greenfields that will be asked for information. The agent will then select the plot

Table 7.1: Agents types and their preferences. Average values. (Source : own research)

Type	HEALTH	EDUCATION	SPORTS	CULTURAL	FOOD	AFFORD
ELDERLIES	1	0.2	0	0.7	0.8	1800€
FAMILIES	0.9	1	0.7	0.3	0.5	1750€
SINGLES	0.2	0.2	1	1	0.8	1700€

that maximises the following function f:

$$f(a,d) := \begin{cases} -1 & \text{if plot price > agent budget} \\ \sum_{j \in J} \frac{a_{ij}}{d_j} & \text{in other case,} \end{cases}$$

where a_i is the preference vector of the agent i, d_j is the distance from each building to the infrastructure j (see Table 7.1) and J are the categories in Table 7.2. Next, the agent will try to buy this plot. In case some other agent has already bought it, the current agent will try to acquire its next preferred option until the plots reach the minimum desired quality set. Please note that it may be possible for an agent not to get a greenfield. Finally, the electrical load generated by each agent is added to the corresponding electrical infrastructure following the function l:

$$l(a,d) := E_t + I_a \cdot S_t \cdot A \cdot P_c,$$

where E is the previous load in that particular electrical infrastructure, I_a is the electrical intensity of agent a (i.e. how much power will the new settlement need), S_t is the simultaneity factor of the loads in that particular infrastructure, A is the area covered by this plot, and P_c is the power intensity of the area, measured as:

$$P_c := \frac{|B_{300}| \sum_{c \in C_{300}} p_c}{|C_{300}| \sum_{b \in B_{300}} s_b},$$

where B_{300} is the set of buildings within 300 meter radius, C_{300} is the set of clients within a 300 meters radius, $|\cdot|$ denotes the set cardinality, p_c is the contracted power

by client c, and s_b is the total surface of the building (measured as the constructed area times the floor count).

By combining these data, when an agent is assigned to an available greenfield, the system analyses the consumption of the neighbouring parcels and size of the buildings in order to predict how much power this new settlement will need. This amount is then added to the total load of the transformer from which the plot feeds.

7.2 Modelling physical reality

Physical and electric elements involved in long term load forecasting were modelled into the simulation for agents to be able to query, select the best greenfields that suited their preferences and connect their settlements to the power network.

7.2.1 Substations and transformers

Substations and transformers are modelled as Passive Entities as their only function is to receive the power demands of the new households (see **??** below for more details).

Each transformer t and substation s has a nominal power, a demanded power (E_t and E_s) and a simultaneity factor (S_t and S_s). Since the electric grid and its elements need to size their nominal power in order to manage demand peaks, the simultaneity factor is critical. It is calculated comparing the maximum power demand with the contracted power of the clients. In that way, it adjusts the theoretical total consumption of the clients to real conditions. Unfortunately, the dataset did not include power measurements for the transformers. In fact, the lowest level where real measures were accessible was at substation outputs. Therefore, the research had to calculate the simultaneity factor S_t at this level, inheriting the simultaneity factor to all the transformers connected to it. The formula used is:

$$S_t := \frac{r_s}{\sum_{t \in T_s} \sum_{c \in C_t} p_c},$$

where r_s denotes the maximum measure registered at the substations output s, T_s denotes the set of transformers connected to that substation output, C_t denotes the set of clients connected to transformer t and finally p_c denotes the contracted power of client c.

7.2.2 **Buildings**

Buildings are those residential dwellings situated in the environment. Every building has its physical representation, concerning location, area, building levels, and the electrical information regarding the amount of clients, the transformer they are connected to and how much power they demand. Buildings are modelled as Passive Entities, so that greenfields (Autonomous Agents) can ask them in order to forecast which type of construction they will harbour.

7.2.3 **Greenfields**

Greenfields are modelled as agents that emulate the land plots where new buildings can be constructed. Local authorities define a location as *accessible to citizens* if it is within 5 minutes walking. That is, considering that the speed of a pedestrian is 4 km/h, this distance corresponds to 300 meters. Therefore, every greenfield has the skill to perceive its surroundings up to 300 meters. As a first approximation, these agents will search and calculate the straight-line distance d_i to several public facilities (e.g. green zones, public transports, parking spaces, and the like). Table 7.2 shows a comprehensive list of these public facilities.

One of the problems faced when working with developable land use, is that, though not yet urbanised, some of these areas have already been split into smaller parcels while others comprise a whole rural area. Although clipped greenfields give some clues about the type of buildings they may contain, without such information, it is still hard to determine the type of construction and how many citizens will host each greenfield. So, as next step, in order to overcome this problem the greenfield estimates, using a spatial moving window smoothing (Kiesel and Wenkel, 2007), the

type of building, number of dwellings and building levels expected according to all adjacent buildings within 300 meters.

Finally, based on the loaded information, each plot establishes its price per square meter and searches for its nearest transformer substation, which the new settlements will connect to. The process is as follows: first a Voronoi diagram is calculated from the set of transformers that serve more than one client (single-client transformers are owned individually, which means that they are not accessible by the DSO). Then, the plot makes the connection with the transformer whose area of influence presents the largest intersection with its own surface.

Table 7.2: Factors considered. (Source : own research)

Factor	Infrastructures considered
HEALTH	Hospitals, clinics
EDUCATION	Schools, colleges, kindergartens, universities
SPORTS	Public swimming pools, pitch, stadiums
CULTURAL	Art centres, theatres, community centres, conference centres, museums, libraries, cinemas
FOOD SHOPS	Food and convenience shops, department stores, supermarkets

7.3 Results of the Geosimulation

For each independent year which electrical growth, the validation process creates a historical map that describes the status of the buildings of the city. In addition, it creates several distributions of agents in order to contrast the process outcome with the real settlements:

Environment: The validation process identifies the buildings that were registered as such from a given year onward and marks them as available greenfields, considering the date in which the building was created is the same as the oldest settlement registered.

Agent System: All the scenarios will create the same amount of agents as new possible settlements that may appeared on that year. The main difference will be the amount of agents of each type created at each case.

Greediness: I have validated the system considering agents with and without greediness. Obviously, the experiments that do not consider agent's greediness obtain better results as the agents are aware of the whole environment. However, the experiments where the greediness is considered are provide as it is a more realistic scenario.

Plots' price: The unitary price of the plots is estimated using recent appraisals performed by the appraiser Tasamadrid.

I defined different metrics in order to test the results obtained with this model. The error can be split into two categories: spatial errors and effective errors.

Spatial Errors: Errors related to the spatial component of the forecast. Namely, how many agents correctly select the year y in which a greenfield has been built ($hits_y$), how many agents incorrectly select the year y in which a greenfield has been built ($semi_y$) and how many agents have completely failed by selecting a greenfield that even today has not been built ($fails_y$). Thus, it is measured as:

$$hits_y := \frac{a_y}{b_y}, \qquad semi_y := \frac{f_y}{\sum_{j=y+1}^{2010} b_j}, \qquad fails_y := 1 - hits,$$

where a_y denotes the number of agents that correctly select a plot that is built in the year y, f_y denotes the number of agents that incorrectly set on a plot in the year y which in reality was built in the following years, b_y denotes the number of plots that have indeed been built in the year y. The variables $hits$, $semi$ and $fails$ are just mean values over of $hits_y$, $semi_y$ and $fail_y$ respectively.

Effective Errors: In this category the load forecasting error is measured. Traditionally, this error is calculated using the MAPE error (Hyndman and Koehler, 2006):

$$mape := \frac{1}{6|S|} \sum_{s \in S} \sum_{y=2005}^{2010} \frac{|r_y^s - p_y^s|}{r_y^s},$$

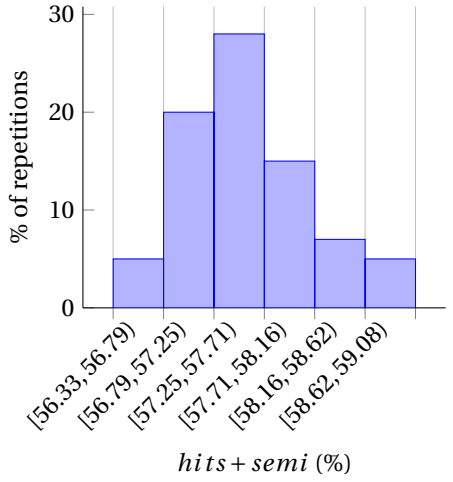
where S denotes the set of transformers, $|\cdot|$ denotes the set cardinality operator, p_y^s denotes the model's forecast for the maximum load of substation s at year y and r_y^s denotes the real one.

Please note that, in this model, MAPE is not a reliable source of error as there is a large variability among the experiments. This situation is due to technical or electrical infrastructure exploitation issues that force a building to not be always connected to its nearest transformer. In order to mitigate this problem, a possibility solution is to measure the error one level higher in the electrical infrastructure, that is, at substation level. However, this measurement would blur the results and hinder the possibility to identify zones where the model is not working correctly. Moreover, in terms of performance, the calculation of this value is quite complex due to the number of database queries that would need to be performed which, in turn, increases the execution time of each simulation.

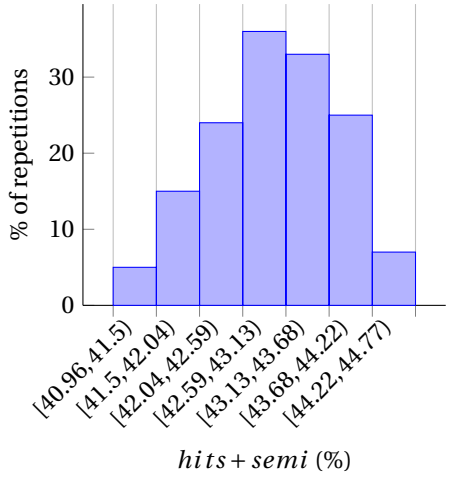
The forecasting ability of the model in two situations was also tested: for one year ahead and five years ahead forecasts. Moreover, as a contrast method I used random assignation of agents to greenfields. In order to calculate an approximated p-value I used a Monte Carlo approach. Namely, I have repeated 100 times the random assignation for every year to approximate the density function of the random variable $hits + semi$ under the null hypothesis that our method and the random Monte Carlo approach would derive the same results. From the approximate density function I calculate the associated $p - value$ (see $pval$ in Table 7.3)

In the first case, each year is evaluated independently, and the outcome is not considered for the following evaluation. Figure 7.1(a) shows the empirical probability density function constructed with 100 repetitions of the contrast method. Table 7.3 shows the numerical results for 1 year ahead forecast (mean value of the 5 years). The first three columns represent the percentage of agent of every type created in that particular experiment. Please note that the 0/0/0 row represents the Monte Carlo approach.

On the other hand, for the 5 year forecasting, I used a *rolling forecast* approach (Hyndman and Athanasopoulos, 2014). Namely, I use the forecast results of the previous years to predict the following year. As for the 1 year ahead case, Figure 7.1(b) shows the empirical probability density function constructed with 100 repetition of the contrast method and Table 7.3 shows the numerical results of the five years ahead forecast.



(a) 1 year ahead forecast



(b) 5 years ahead forecast

Figure 7.1: Probability density function of the random variable $hits + semi$. (Source : own research)

Table 7.3: Experimental results for the different agent type composition. All measures are in %.

			1 year forecast				5 years forecast			
Elderly	Families	Young	$hits$	$semi$	$fails$	$pval$	$hits$	$semi$	$fails$	$pval$
0	0	0	0.14	0.44	0.42	0.41	0.18	0.25	0.57	0.45
100	0	0	0.17	0.49	0.34	0.00	0.34	0.31	0.35	0.00
0	100	0	0.22	0.50	0.28	0.00	0.34	0.35	0.32	0.00
0	0	100	0.19	0.55	0.26	0.00	0.29	0.37	0.34	0.00
50	25	25	0.17	0.51	0.31	0.00	0.30	0.35	0.35	0.00
25	50	25	0.21	0.49	0.30	0.00	0.33	0.34	0.34	0.00
25	25	50	0.17	0.55	0.28	0.00	0.27	0.37	0.36	0.00
80	10	10	0.18	0.49	0.33	0.00	0.34	0.31	0.35	0.00
10	80	10	0.22	0.49	0.29	0.00	0.34	0.35	0.32	0.00
10	10	80	0.19	0.55	0.26	0.00	0.29	0.36	0.34	0.00
60	20	20	0.17	0.50	0.33	0.00	0.34	0.32	0.35	0.00
20	60	20	0.21	0.49	0.30	0.00	0.33	0.34	0.33	0.00
20	20	60	0.18	0.54	0.28	0.00	0.28	0.37	0.35	0.00

Figure 7.1 shows, as expected, a large difference between the errors of the experiment in the 1 year ahead forecasting and the 5 years ahead forecasting. Not only do the results of the random assignation are better in the 1 year ahead forecasting, but they also have considerably less variance. Table 7.3 confirms this expectation. In the case of the 1 year ahead forecasting the mean value of $hits + semi$ is around 70 %, while in the case of 5 years forecasting it is around 66 %. Please note that this value is quite high considering that the random assignation has achieved 58 and 43 % (respectively) in those experiments. The results of using the proposed MAS model are in all cases significantly better than those corresponding to the random Monte Carlo approach. Take into account that the first row corresponds to the Monte Carlo approach, so in this particular case, the same results are being compared.

The distribution of $hits$ and $semi$ in both experiments is also interesting. While in the 1 year ahead forecasting experiment $semi$ is quite high (around 50 %) in

the 5 years ahead forecasting experiment is near 30 %. The explanation of these results is straightforward: in the 1 year ahead forecasting experiment, the number of greenfields that (if occupied) score a $semi$ decrease with every year, while in the case of 5 years ahead forecasting they are constant. Please note that it is impossible to score a $semi$ in the last year of the 1 year ahead forecasting experiment since I do not have information of the next year and all the houses of previous years have already been occupied.

The results show how different agent type distributions affect the accuracy of the prediction delivered by our model. As the typical house buyers in Spain are families, it is expected that models with a high percentage of families agents will perform better. The results from the 5 years ahead forecasting confirm that hypothesis. The best agent mix consists of considering only agents of Family type while in the mixed cases, those with high percentage of Family agents perform at least as well as the other. On the contrary, it seems that in the short term (1 year ahead forecasting), the Young are the main variable. This can be explained as Young frequently change their homes, so in the short term, buildings near their favourite places are occupied in a short time.

Qualitatively, it can be seen that the layout follows the logic imposed by the vector of preferences in the settlement of the agents. Figures 7.2 to 7.4 show the distribution of Elderly, Families and Young agents through three different heatmaps. The first one shows how the final distribution of the Elderly and the localisation of Clinics and Hospitals. In the second one, I present the final distribution of Families and the localisation of Schools. Finally, in the third one, I present the final distribution of Young agents and the localisation of Sport Centres. In the three cases the final distribution loosely approximate to the localisation of the corresponding infrastructures.

Figure 7.2: Heatmap of the new settlement for Elderly people and Hospitals.

Figure 7.3: Heatmap of the new settlement for Families and Schools.

Figure 7.4: Heatmap of the new settlement for Young People and Sports Centres.

7.4 Conclusions and Future work

The methodology described and its implementation in the GeoWorldSim framework has achieved successful results in the urban Long Term Load Forecasting process. By Geosimulating the social preferences that affect house buying with dynamic offer and demand, the success rate when issuing 1 year ahead forecast is above 70 % and 66 % when issuing a 5 years ahead forecasting. These results allow to obtain a better representation of how our urban demography will change and the resilience of the electrical network to it. Additionally, the analysed problem emphasises some research challenges that are important for the further exploration of spatial load forecasting within Geosimulations.

- New smart power grids demand a huge effort in redesigning and enhancing current power networks, as well as integrating distributed generation, renewable energies and the electric vehicle. Contrary to the previous centralised and unidirectional model (i.e., from centralised generation via the transmission

and distribution grids to the customers), smart girds will change to a bidirectional power flow model that will include distributed generation (mainly from renewable sources) and the electric vehicle.

- The power network operation needs to be safe, reliable and, as long as possible, cost effective. The new scenario enabled by the massive adoption of distributed generation and storage and the intensive application of information technologies render the old centralised and static architecture unfeasible. The main reason is that gathering the complete information to achieve optimal management, if possible at all, would entail a prohibitive cost in infrastructure and time. Therefore, a trade-off must be found between distributing the intelligence and the costs associated to this decision.
- The grid computing paradigm presents inspiring characteristics for modelling agent environments that will help the smart grid vision come true. Indeed, grid computing applications are based on a group of distributed nodes carrying out a task coordinated by a central entity: power networks do have many nodes (e.g., meters and substations) and tasks that require participation and coordination (e.g., invoicing: demands retrieving the consumption data from all clients, applying the corresponding prices, and notifying the clients) among all of the nodes. MAS and agent environments can tackle this challenge by distributing the intelligence all over the grid by means of individual intelligent agents controlling a number of assets.
- The management and configuration of power grids is closely linked with urban evolution. Social dynamics dictate the appearance of new settlements as well as the amount of energy demanded depending on the economic availability, the age and preferences of new costumers and the own evolution of the city. The edification of new infrastructures such as hospitals, schools, parks, malls; even the construction of new roads heavily influence the citizens' decisions and mobility.

CHAPTER

8

Conclusions and Future

The dissertation presented hereby has described what Geosimulations are and why this paradigm is starting to gain prominence in diverse areas. Although MAS and GIS technologies are well established tools in the research community and industry there are still many urban analyses in which these have not been still introduced. This is the reason why this research set the goal of proving that using Geosimulations and empowering them with more proactive, distributed and asynchronous Agent Environments would allow a better description of urban scenarios. This closing chapter contains the lessons learned through the research, summarises the limitations found and sketches some future lines of work.

8.1 Achievements

Throughout all the research process, several contributions have been made in the Geosimulation field, relying on previous works and proposing some new mechanisms in order to enhance the processes involved. To this end, a new framework for Geosim-

ulation, GeoWorldSim, has been presented in Chapters 3 and 4 which contributes to improve the current state-of-the-art. GeoWorldSim simulation core manages to put into practice the latest theories in MAS along with the new techniques in cloud computing and asynchronism. Below is shown a list with the tasks and challenges addressed, together with the designed approaches:

- Design of a new general purpose Geosimulation framework based on C++ and cloud computing.
- Environment oriented with Environments that are first class proactive entities.
- Design of the software architecture for the framework in 9 enabler modules that facilitate GeoWorldSim's deployment in the cloud paradigm. The architecture provides:
  - JSON-LD compliant platform to serialise and deserialise any type entity.
  - Decoupling of the simulation and rendering.
  - Use of asynchronous message passing in all the framework to lighten up the performance.
  - Distributed simulations.
  - A novel transparent mechanism for distributed simulations synchronisation.
  - Real time monitor and interactions with the simulations.
  - A visual logic composer module for none experts to build complex intelligence.
  - Seamless integration with IoT frameworks, such as FIWARE (Consortium, 2016), LinkSmart (Souza and Amazonas, 2013) and Sentilo (Bain, 2014).
- Integration with third party domain models such as Matlab, R, EPASWMM, EPA-NET.

Parallel to the development of the simulation core, GeoWorldSim was split in ten modules to build resilient cloud the architecture. In order to achieve effective Geosimulations, it was decided to focus the core on only carrying out the evolution of a scenario and delegate data transformations and visualisation in secondary modules.

GeoWorldSim has become a platform to address the needs of Geosimulations meeting the characteristics contained in Table 8.1.

	Functionality
Open Source	✓
Programming language	JSON-LD, C++
FIPA messaging	✗
Large-scale	✓
Environment centric	✓
Standard ontology based	✓
Asynchronous coordination	✓
GIS Integrated	✓
Checkpoints	1
Distributed simulations	✓
Simulation GUI	✓
Visual programming	✓
Charts	✓
Web based	✓
Multiple users	✓

Table 8.1: GeoWorldSim characteristics. (Source: own research)

Once the platform was built, it was tested and extended in the three use-cases described in Chapters 5 to 7. Long Term Load Forecasting use-case, Chapter 7, became the starting point for the development of a Geosimulation platform. It required going further than traditional GIS analyses and developing a housing search competitive MAS with geographic capabilities. The contributions here have been:

- Deep analysis of the state-of-the-art in Geosimulations.
- Foundation of GeoWorldSim and implementation of its primitives *Agents*, *Environments*, *Behaviours* and *Skills*.
- Geosimulation of the factors that affect house buying with dynamic offer and demand.
- Geosimulating how our urban demography will change and the resilience of the electrical network to it.

- Success rate above 70 % when issuing 1 year ahead forecast and 66 % when issuing a 5 years ahead forecasting.
- Best demo award in the AAMAS (International Foundation for Autonomous Agents and Multiagent Systems (IFAAMAS), 2014) 2014 Demo track.

Green Travelling, Chapter 6, meant an extension of GeoWorldSim in the transportation modelling area. It contributed with a deep analysing and implementation of an entire four step transportation model. This is the traditional behaviour all travellers follow when commuting from house to work including the selection of mode of transport and performing the travel itself. The contributions here have been:

- Extend GeoWorldSim with the Network Environment capabilities, *Graph* elements and *Routing* models.
- Adding a Fuzzy Logic model for agents to use in their decision processes.
- Deep analysis and comparison of eight data science methods for transport choice modelling.
- Geosimulation of citizens transport choice in their everyday commuting and travel execution.
- Geosimulating the impact of applying green travelling policies to urban travels.
- Success rate about 45 % for plain Expert Knowledge and above 50 % when real data to train models for transport choice forecasting.

Ubiquitous Smart Cities, Chapter 5, entailed the design of a novel Human Behaviour model and a test-bed for measuring the impact of Smart City infrastructures. The developed Behaviour model was then set up in GeoWorldSim as its mechanism for Behaviours construction. The Geosimulation achieved:

- Extend GeoWorldSim with a novel Human Behaviour model.
- Geosimulating the impact IoT infrastructures have in network communications.
- Success rate between 88 % and 94 % for emulation of Behaviour and 98 % and 99 % for emulation of Behaviour duration.

Therefore, it can be concluded that the research hypothesis (please refer to section 1.5) is validated, as it has been demonstrated that the designed Geosimulations have achieved better modelling of the studied Urban Systems. All the research emphasises the crucial role Geosimulations have in evaluating urban scenarios and proves that there are many areas in which GeoWorldSim has a great potential to improve traditional analyses.

8.2 Discussion and Limitations

Research is an iterative process that can never be thought of as a concluded task. Obstacles found throughout the process may require a turn in the methodology and advances made always lead to the emergence of new questions. One of the most recurrent question always emerges is: if going back to the beginning of this research would it have taken the same decisions and assumptions? In this sense, the main questions and limitations found have been:

8.2.1 Create or reuse

Long has been discussed about whether this effort and research should have been used to improve some of the existing platforms instead of developing a new one from scratch. Furthermore, taking into consideration how the GeoWorldSim framework has finally been modularised, nothing prevented from using one existing platform as simulation core and just implementing the drivers to connect it to the rest of the modules. In chapter 2 a revision of the existing platforms was made as of 2013 and their development status at that time. At that time it was decided to leave them aside and develop an own framework in which the hypothesis presented hereby could be validated and whose problems would try to solve. This scope toke some assumptions that guided and constrained the research approach.

While this research had to struggle with basic programming issues, state-of-the-art platforms had said problems already solved and were being improved with the identified as missing. Probably at present with the current evolution of the

existing platforms, this research would have opted to use Gama (Taillandier et al., 2010) as a simulation core and develop additional external modules as the ones in GeoWorldSim.

It is impossible to know if that change in the research would have led to a dead end regarding implementing the required functionalities in Gama platform for several reasons:

- Gama's internal simulation engine may require a complete refactor not allow to implement the desired functions.
- Gama developer and user community could find the desired implementations out of the scope of the platform and not willing to include them.
- The performance obtained once implemented could not be as good as expected.

Whatever could have been, the decision was made to create an own framework with full control of the code and the obtained results, though at an early stage, suggest that GeoWorldSim is promising for becoming a great tool to compete with current platforms.

8.2.2 **Data-sets**

In every research, information plays a very important role and obtaining reliable GIS and social data is often the most difficult and costly part of a Geosimulation. Results obtained in this dissertation have been as good as the data it was possible work with. Not only when designing the simulation but also in order to calibrate and further test how good the model is. GeoWorldSIm framework could be extended to cover new use-cases in urban evaluations if data can be found to model the Agents and Environments.

Although authorities have opened many open data repositories many are outdated or lacking in detail which makes necessary to look for other sources of information. In this sense, some communities have collected their own specific data in initiatives such as OpenStreetMap (OSM). The OSM project aims at creating an open

source world map based on volunteer geographic information. The OSM community is in charge of constantly keeping the data-set up to date and creating tools to operate said data. Similar collaborative initiatives in other areas (e.g. consumption patterns, smart metering data, traffic data) would greatly expand the range of Geosimulations that could be built.

8.3 Future work and open issues

Geosimulations have a bright future ahead. This research believes that their destiny is not only to be implemented in decision making systems for experts (e.g. technicians, policy makers, consultancy) but into everyday services' background citizens use recurrently (e.g. transport, shopping, leisure). While the above use-cases have focused in evaluating *what-if* scenarios, the goal of a Geosimulation can also be to run in parallel and in sync with the real world providing services to real world users. Emerging technologies such as wireless sensor networks and the IoT provide a whole new virtual layer where the physical world can be accessed or modified by any computational system. This information gathering capabilities from distributed sources is poised to revolutionise the way MASs environments interact with the real world by putting humans *in the loop*. Furthermore, artificial intelligence will transform everyday objects into pro-active actors. Geosimulations appear as an effective approach for modelling and designing the systems that give answer to user needs through these active objects.

Future research should make efforts in designing these *live* Geosimulations for situations where it will be necessary that real-world humans become an integral part of the system and live together within the same virtual environment with intelligent agents. This is one of the open challenges that emerges when bringing urban planning simulations to the limit. A new type of agent environments that henceforth will be referred to as *mixed environments* is proposed to realise mixed multi-agent systems populated by both human and artificial agents.

Mixed environments need to face complex decisions in to how model humans, design a human-aware communication infrastructure, provide decision and co-

ordination support, and how to implement laws. Open challenges in that path include:

- Devising effective environment mechanisms effectively modelling human-agent interaction and coordination, paying particular attention to providing decision and coordination support;
- The introduction of high-level environment abstractions and mechanism to effectively model and design *augmented* realities and worlds where (human/software) agents live in;
- Methodologies for designing and developing scalable agent environment technology stacks to deal with open, large-scale human-agent environments, eventually integrating mainstream architectures and technologies related to e.g. Internet-of-Things and the cloud.

This new line of research (Ricci et al., 2015a) was identified during the course of E4MAS (Weyns, 2014) held at the 13th International Conference on Autonomous Agents and Multiagent Systems (International Foundation for Autonomous Agents and Multiagent Systems (IFAAMAS), 2014).

In this sense, GeoWorldSim has been prepared to face this challenge by adding some flags and features to run in *live* mode. Currently, *non-live* simulations would finish their execution once all agents have finished performing and achieved their goals. However, running GeoWorldSim in *live* mode makes the Execution and Time Environments stick to real world's date, time and speeds in a continuous execution that has no end. Using the infrastructure created for distributed simulations, a *live* Geosimulation can receive real world human information just as if it were from another distributed instance of GeoWorldSim. This mechanism enables:

- Keeping real world passive entities (e.g. roads, waste deposit points, water networks) information updated from sensor data. The Messaging Environment would listen for changes in entity attributes and propagate them to each of the entities in the Geosimulation.
- Creating on-the-fly intelligent agents that are born inside the Geosimulation on demand to act as *virtual assistants* whenever a real world user requires a service

to be provided. Through the Messaging Environment a complete agent, with its behaviour and skills, can be created from any JSON-LD compliant message created by an IoT device. This assistant would be registered in all environments and executed in order to calculate the desired result for the user.

Bringing together all the *virtual assistants* in a common Geosimulation rather than isolated calculations, would allow to be able to put into cooperation or competition these assistant agents. Centralising the current state of the real world together with the queries of what users want to perform allow knowing in advance changes that will happen in that real world state. Some interesting applications could be:

Scheduling applications: Current itinerary scheduling systems, such as the ones for traffic management, use real data together with forecasts to *route* traffic through different paths according to real or expected congestion. However, since the routes are calculated in isolated requests, all people asking for a route at the same time would receive the same itinerary, resulting in subsequent traffic congestion. Having an active Geosimulation with *virtual assistants* can greatly improve the service. Requesting big amounts of itineraries for the same time would make the *virtual assistants* compete with each other in a shared environment provisioning roads, public transport seats or even space in the sidewalk and therefore returning different routes avoiding sending all users through the same itinerary.

Resources procurement: Many of the collaborative economy tools are based on putting people in contact so that they can collaborate for getting a better purchase of some resource such as energy auctions, joint holiday planning or circular economy initiatives. The scope of these tools usually ends when the users get in touch and it is up to them to negotiate and agree on the final decision. MASs have been modelling autonomous auction and cooperation processes for years (Parsons et al., 2011) what can improve these collaborative platforms with capabilities of delegating users decisions into these *virtual assistants*.

Existing explorations and visions about these *mixed environments* and *assistant agents* can be considered just as a starting point and the way forward to bring Geosimulations to a general public.

8.4 Final personal remarks

The presented work, is the result of a five year long pre-doctoral period. This research arises from the personal concerns of the doctoral student aligned with the objectives and tasks it has been developing in DeustoTech's Energy and Environment Unit (de Deusto, a,b) at the University of Deusto. These five year research is expected to contribute in a broader use and understanding of MASs and especially Geosimulations. This is a research field that the doctoral student considers of great relevance for the decision making in physical and organisational aspects of any process in which citizens or individuals are involved. Within the Energy and Environment Unit, all this knowledge will be adopted and GeoWorldSim will become one of its core tools. Similarly, it is wished that the extracted conclusions, together with the drafted future lines of work will inspire other researchers to develop their own ideas, contributing to the subject and improving the world of Geosimulations.

Bibliography

Abar, S., Theodoropoulos, G. K., Lemarinier, P., and OâĂŹHare, G. M. (2017). Agent based modelling and simulation tools: A review of the state-of-art software. *Computer Science Review*, 24:13–33.

Adelantado, F., Vilajosana, X., Tuset-Peiró, P., Martínez, B., and Melià, J. (2016). Understanding the limits of lorawan. *CoRR*, abs/1607.08011.

Agafonkin, V. Leafletjs.

Al-Bahadili, H. (2012). *Simulation in Computer Network Design and Modeling: Use and Analysis*. IGI Global.

Albrecht, J. (2005). A new age for geosimulation. *Transactions in GIS*, 9(4):451–454.

Alfares, H. K. and Nazeeruddin, M. (2002). Electric load forecasting: literature survey and classification of methods. *International Journal of Systems Science*, 33(1):23–34.

Allen, A. O. (2014). *Probability, statistics, and queueing theory*. Academic Press.

Altman, N. S. (1992). An introduction to kernel and nearest-neighbor nonparametric regression. *The American Statistician*, 46(3):175–185.

Amir, Z. (2014 (accessed July 9, 2018)). Qhttp a light-weight and asynchronous HTTP library (both server & client) in Qt5 and C++14.

Antrim, A., Barbeau, S. J., et al. (2013). The many uses of gtfs data–opening the door to transit and multimodal applications. *Location-Aware Information Systems Laboratory at the University of South Florida*, 4.

Appel, A. W. and Jim, T. (1989). Continuation-passing, closure-passing style. In *Proceedings of the 16th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 293–302. ACM.

Atkins Jr, T., Etemad, E. J., and Atanassov, R. (2013). Css grid layout.

Awaludin, A. and Chen, D. (2007). Urbansim parallel programming - capstone paper.

Azkune, G., Almeida, A., López-de Ipiña, D., and Chen, L. (2015). Combining Users' Activity Survey and Simulators to Evaluate Human Activity Recognition Systems. *Sensors*, 15(4):8192–8213.

Bahu, J.-M., Koch, A., Kremers, E., and Murshed, S. M. (2014). Towards a 3d spatial urban energy modelling approach. *International Journal of 3-D Information Modeling (IJ3DIM)*, 3(3):1–16.

Bain, M. (2014). Sentilo-sensor and actuator platform for smart cities. *Retr. Febr*, 20:2015.

Barbosa, J. and Leitão, P. (2011). Simulation of multi-agent manufacturing systems using agent-based modelling platforms. In *Industrial Informatics (INDIN), 2011 9th IEEE International Conference on*, pages 477–482. IEEE.

Batty, M. (1999). Multi-agent approaches to urban development dynamics. *Research proposal for the Environmental and Social Research Council. Centre for Advanced Spatial Analysis, University College London. London.*

Bazzan, A. L. C. and Klügl, F. (2014). A review on agent-based technology for traffic and transportation. *The Knowledge Engineering Review*, 29(3):375âĂŞ403.

Beazley, D. (2010). Understanding the python gil. In *PyCON Python Conference. Atlanta, Georgia.*

Behnisch, M. and Meinel, G. (2018). *Trends in Spatial Analysis and Modelling*. Springer.

Bellifemine, F., Bergenti, F., Caire, G., and Poggi, A. (2018 (accessed July 1, 2018)). Jade behaviour class.

Bellifemine, F., Poggi, A., and Rimassa, G. (1999). Jade–a fipa-compliant agent framework. In *Proceedings of PAAM*, volume 99, page 33. London.

Benenson, I., Aronovich, S., and Noam, S. (2001). Obeus: Object-based environment for urban simulations. In *Proceedings of the Sixth International Conference on GeoComputation*.

Benenson, I., Torrens, P. M., and Torrens, P. (2004). *Geosimulation: Automata-based modeling of urban phenomena*. John Wiley & Sons.

Benjamini, Y. (1988). Opening the box of a boxplot. *The American Statistician*, 42(4):257–262.

Berkan, R. C. and Trubatch, S. (1997). *Fuzzy system design principles*. Wiley-IEEE Press.

Bizer, C., Heath, T., and Berners-Lee, T. (2009). Linked data-the story so far. *International journal on semantic web and information systems*, 5(3):1–22.

Blecic, I., Borruso, A., Cecchini, A., DâĂŹArgenio, A., Montagnino, F., and Trunfio, G. A. (2009). A cellular automata-ready gis infrastructure for geosimulation and territorial analysis. In *International Conference on Computational Science and Its Applications*, pages 313–327. Springer.

Blecic, I., Cecchini, A., and Trunfio, G. A. (2008). A software infrastructure for multi-agent geosimulation applications. In *International Conference on Computational Science and Its Applications*, pages 375–388. Springer.

Borges, C., Penya, Y., and Pijoan, A. (2012). Agent based spatial load forecasting. In *Proceedings of 3rd International Workshop on Agent Technologies for Energy Systems*

(ATES 2012) held at the 11th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2012), pages 107–108.

Borges, C. E., Kamara Esteban, O., Pijoan, A., and Penya, Y. K. (2014). Multi-agent gis system for improved spatial load forecasting. In *Proceedings of the 2014 international conference on Autonomous agents and multi-agent systems*, pages 1667–1668. International Foundation for Autonomous Agents and Multiagent Systems.

Borges, C. E., Pijoan, A., Sorrosal, G., Oribe-Garcia, I., González, M., and Kamara Esteban, O. (2013). Uso de fuentes de información geográfica voluntarias en proyectos de ingeniería.

Borshchev, A., Karpov, Y., and Kharitonov, V. (2002). Distributed simulation of hybrid systems with anylogic and hla. *Future Generation Computer Systems*, 18(6):829–839.

Bouchard, K., Ajroud, A., Bouchard, B., and Bouzouane, A. (2010). *Advances in Computer Science and Information Technology*, volume 6059 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, Berlin, Heidelberg.

Box, G. E. (1979). All models are wrong, but some are useful. *Robustness in Statistics*, 202.

Bruneau, J., Jouve, W., and Consel, C. (2009). DiaSim: A parameterized simulator for pervasive computing applications. In *Mobile and Ubiquitous Systems: Networking & Services, MobiQuitous, 2009. MobiQuitous' 09. 6th Annual International*, pages 1–10. IEEE.

Buchmayr, M., Kurschl, W., and Küng, J. (2011). A Simulator for Generating and Visualizing Sensor Data for Ambient Intelligence Environments. *Procedia Computer Science*, 5:90–97.

Carcellar, B. G. (2017). Perception modelling of visitors in vargas museum using agent-based simulation and visibility analysis. *International Archives of the Photogrammetry, Remote Sensing & Spatial Information Sciences*, 42.

Casado-Mansilla, D., Moschos, I., Kamara-Esteban, O., Tsolakis, A. C., Borges, C. E., Krinidis, S., Irizar-Arrieta, A., Kitsikoudis, K., Pijoan, A., Tzovaras, D., et al. (2018). A human-centric & context-aware iot framework for enhancing energy efficiency in buildings of public use. *IEEE Access*.

Center for International Earth Science Information Network - Columbia University, United Nations Food and Agriculture Programme, and Centro Internacional de Agricultura Tropical (2014). Gridded population of the world, version 3 (gpwv3): Population count grid. palisades, ny: Nasa socioeconomic data and applications center (sedac).

Chang, C.-C. and Lin, C.-J. (2004). C.-j.: Libsvm: a library for support vector machines.

Chang, C.-C. and Lin, C.-J. (2011). Libsvm: A library for support vector machines. *ACM Trans. Intell. Syst. Technol.*, 2(3):27:1–27:27.

Change, P. L.-U. (2000). A summary of models for assessing the effects of community growth and change on land-use patterns. *US Environmental Protection Agency, Office of Research and Development, Cincinnati, Ohio*.

Chen, L. (2012). Agent-based modeling in urban and architectural research: A brief literature review. *Frontiers of Architectural Research*, 1(2):166–177.

Chen, L., Hoey, J., Nugent, C., Cook, D., and Yu, Z. (2012). Sensor-based activity recognition. *IEEE Transactions on Systems, Man, and Cybernetics-Part C*, 42(6):790–808.

Chen, L. and Khalil, I. (2011). Activity recognition: Approaches, practices and trends. In *Activity Recognition in Pervasive Intelligent Environments*, pages 1–31. Springer.

Chen, Y. and Wang, J. Z. (2003). Support vector learning for fuzzy rule-based classification systems. *IEEE Transactions on Fuzzy Systems*, 11(6):716–728.

Chiu Chuen, O., Karim, M. R., and Yusoff, S. (2014). Mode choice between private and public transport in klang valley, malaysia. *The Scientific World Journal*.

Claridades, A. R. C., Villanueva, J. K. S., and Macatulad, E. G. (2016). Evacuation simulation in kalayaan residence hall, up diliman using gama simulation software. *International Archives of the Photogrammetry, Remote Sensing & Spatial Information Sciences*, 42.

Clauss, T. and Döppe, S. (2016). Why do urban travelers select multimodal travel options: A repertory grid analysis. *Transportation Research Part A: Policy and Practice*, 93:93–116.

Collier, N. and North, M. (2013). Parallel agent-based simulation with repast for high performance computing. *Simulation*, 89(10):1215–1235.

COM-EU-Commission et al. (2007). Results of the review of the community strategy to reduce co2 emissions from passenger cars and light-commercial vehicles: Impact assessment, february 7, 2007. *SEC (2007)*, 60.

Consortium, F.-W. (2012–2016). FI-WARE architecture. `http://cordis.europa.eu/fp7/ict/netinnovation/deliverables/fi-ware/fi-ware-d231b.pdf`.

Consortium, O. G. et al. (2003). Simple features specification for sql. *Estándar publicado por Open GIS Consortium. Disponible en Internet: http://www. opengis. org/techno/specs/99-049. pdf. Ultima visita*, 30(7).

Consortium, W. W. W. et al. (2014). Json-ld 1.0: a json-based serialization for linked data.

Cook, D. J. and Schmitter-Edgecombe, M. (2009). Assessing the quality of activities in a smart environment. *Methods of information in medicine*, 48(5):480–5.

Council, B. C. (2012). Plan de acción de energía sostenible/sustainable energy action plan.

Cristianini, N. and Shawe-Taylor, J. (2000). *An introduction to support vector machines and other kernel-based learning methods*. Cambridge university press.

Crooks, A., Castle, C., and Batty, M. (2008a). Key challenges in agent-based modelling for geo-spatial simulation. *Computers, Environment and Urban Systems*, 32(6):417–430.

Crooks, A., Castle, C., and Batty, M. (2008b). Key challenges in agent-based modelling for geo-spatial simulation. *Computers, Environment and Urban Systems*, 32:417–30.

Cvetković, B., Janko, V., Romero, A. E., Kafalı, Ö., Stathis, K., and Luštrek, M. (2016). Activity recognition for diabetic patients using a smartphone. *Journal of Medical Systems*, 40(12):256.

Czaplicki, E. and Chong, S. (2013). Asynchronous functional reactive programming for guis. In *ACM SIGPLAN Notices*, volume 48, pages 411–422. ACM.

David, H. (2006). von seggern, crc standard curves and surfaces with mathematica.

Davison, A. C. and Hinkley, D. V. (1997). *Bootstrap Methods and Their Applications*. Cambridge University Press, Cambridge. ISBN 0-521-57391-2.

de Deusto, U. Instituto tecnológico deusto – DeustoTech.

de Deusto, U. Unidad energía y medio ambiente del instituto tecnológico deusto – DeustoTech.

Derrac, J., García, S., Molina, D., and Herrera, F. (2011). A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms. *Swarm and Evolutionary Computation*, 1(1):3 – 18.

Dezső, B., Jüttner, A., and Kovács, P. (2011). Lemon–an open source c++ graph template library. *Electronic Notes in Theoretical Computer Science*, 264(5):23–45.

Dezső, B., Jüttner, A., and Kovács, P. (2011 (accessed June 9, 2018)). Tsp in lemon–an open source c++ graph template library.

Dijkstra, E. (1959). A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271.

Ding, L. and Zhang, N. (2016). A travel mode choice model using individual grouping based on cluster analysis. *Procedia engineering*, 137:786–795.

Domencich, T. A. and McFadden, D. (1975). *Urban travel demand-a behavioral analysis*. North-Holland Publishing Co.

Ecma International (2017 (accessed July 9, 2018)). Ecmascript 2017 language specification.

Efron, B. (1979). Bootstrap methods: Another look at the jackknife. *Ann. Statist.*, 7(1):1–26.

Eugster, P. T., Felber, P. A., Guerraoui, R., and Kermarrec, A.-M. (2003). The many faces of publish/subscribe. *ACM computing surveys (CSUR)*, 35(2):114–131.

Evenden, G. I. (2005). libproj4: A comprehensive library of cartographic projection functions (preliminary draft).

Farahat, M. (2004). Long-term industrial load forecasting and planning using neural networks technique and fuzzy inference method. In *Universities Power Engineering Conference, 2004. UPEC 2004. 39th International*, volume 1, pages 368–372. IEEE.

Fedosejev, A. (2015). *React. js Essentials*. Packt Publishing Ltd.

Fellendorf, M. (1994). Vissim: A microscopic simulation tool to evaluate actuated signal control including bus priority. In *64th Institute of Transportation Engineers Annual Meeting*, pages 1–9. Springer.

Fette, I. and Melnikov, A. (2011). The websocket protocol. Technical report.

Fipa, A. (2002). Fipa acl message structure specification. *Foundation for Intelligent Physical Agents, http://www. fipa. org/specs/fipa00061/SC00061G. html (30.6. 2004)*.

Fischer, C., Olderog, E.-R., and Wehrheim, H. (2001). A csp view on uml-rt structure diagrams. In *International Conference on Fundamental Approaches to Software Engineering*, pages 91–108. Springer.

Fortune, S. (1995). Voronoi diagrams and delaunay triangulations. In *Computing in Euclidean geometry*, pages 225–265. World Scientific.

Furtado, V. and Vasconcelos, E. (2006). A multiagent simulator for teaching police allocation. *AI magazine*, 27(3):63.

Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1993). Design patterns: Abstraction and reuse of object-oriented design. In *European Conference on Object-Oriented Programming*, pages 406–431. Springer.

Gironás, J., Roesner, L. A., Rossman, L. A., and Davis, J. (2010). A new applications manual for the storm water management model (swmm). *Environmental Modelling & Software*, 25(6):813–814.

Green Travelling Consortium (2013). A platform to analyse and foster the use of green travelling options. Technical report, Project Proposal, The ERA-NET Transport III: Future Travelling.

Grignard, A., Taillandier, P., Gaudou, B., Vo, D. A., Huynh, N. Q., and Drogoul, A. (2013). Gama 1.6: Advancing the art of complex agent-based modeling and simulation. In *International Conference on Principles and Practice of Multi-Agent Systems*, pages 117–131. Springer.

Grigoryev, I. (2012). *AnyLogic 6 in three days: a quick course in simulation modeling*. Anylogic North America.

Gruber, T. (1993). What is an ontology. *WWW Site http://www-ksl. stanford. edu/kst/whatis-an-ontology. html (accessed on 07-09-2004)*.

Guerreiro, T. d. C. M., Kirner Providelo, J., Pitombo, C. S., Antonio Rodrigues Ramos, R., and Rodrigues da Silva, A. N. (2018). Data-mining, gis and multicriteria analysis in a comprehensive method for bicycle network planning and design. *International Journal of Sustainable Transportation*, 12(3):179–191.

Guide, M. U. (1998). The mathworks. *Inc., Natick, MA*, 5:333.

Hahmann, S., Burghardt, D., and Weber, B. (2011). "80% of all information is geospatially reference"??? towards a research framework: Using the semantic web for (in) validating this famous geo assertion. In *Proceedings of the 14th AGILE Conference on Geographic Information Science*.

Helal, A., Cho, K., and Lee, W. (2012). 3D modeling and simulation of human activities in smart spaces. In *Ubiquitous Intelligence & Computing and 9th International Conference on Autonomic & Trusted Computing (UIC/ATC), 2012 9th International Conference on*, pages 112–119. IEEE.

Helal, S., Lee, J., and Hossain, S. (2011). Persim-Simulator for human activities in pervasive spaces. In *Intelligent Environments (IE), 2011 7th International Conference on*, pages 192–199. IEEE.

Herrera, F. (2008). Genetic fuzzy systems: taxonomy, current research trends and prospects. *Evolutionary Intelligence*, 1(1):27–46.

Hindriks, K. V., De Boer, F. S., Van der Hoek, W., and Meyer, J.-J. C. (1999). Agent programming in 3apl. *Autonomous Agents and Multi-Agent Systems*, 2(4):357–401.

Hitz, D., Malcolm, M., Lau, J., and Rakitzis, B. (2005). Copy on write file system consistency and block usage. US Patent 6,892,211.

Holovaty, A. and Kaplan-Moss, J. (2009). *The definitive guide to Django: Web development done right*. Apress.

Holtsmark, B. and Skonhoft, A. (2014). The norwegian support and subsidy policy of electric cars. should it be adopted by other countries? *Environmental science & policy*, 42:160–168.

Horni, A., Nagel, K., and Axhausen, K. W. (2016). *The multi-agent transport simulation MATSim*. Ubiquity Press London:.

Hyndman, R. J. and Athanasopoulos, G. (2014). *Forecasting: principles and practice*. OTexts.

Hyndman, R. J. and Koehler, A. B. (2006). Another look at measures of forecast accuracy. *International Journal of Forecasting*, 22(4):679 – 688.

Instituto Nacional de Estadística (2011). Censos 2011. `http://www.ine.es/censos2011_datos/cen11_datos_resultados.html`.

International Foundation for Autonomous Agents and Multiagent Systems (IFAAMAS) (2014). 13th international conference on autonomous agents and multiagent systems.

Jain, N., Bhansali, A., and Mehta, D. (2015). Angularjs: A modern mvc framework in javascript. *International Journal of Global Research in Computer Science (UGC Approved Journal)*, 5(12):17–23.

Jiang, J., Han, G., and Chen, J. (2002). Modelling turning restrictions in traffic networks for vehicle navigation system. *INTERNATIONAL ARCHIVES OF PHOTOGRAMMETRY REMOTE SENSING AND SPATIAL INFORMATION SCIENCES*, 34(4):106–110.

Jordan, R., Birkin, M., and Evans, A. (2010). Agent-based simulation modelling of housing choice and urban regeneration policy. In 11^{th} *International Conference on Multi-Agent-Based Simulation*, pages 152–166.

Kafalı, Ö., Romero, A. E., and Stathis, K. (2014). *Activity Recognition for an Agent-Oriented Personal Health System*. Springer International Publishing, Cham.

Kamara-Esteban, O., Azkune, G., Pijoan, A., Borges, C. E., Alonso-Vicario, A., and López-de Ipiña, D. (2017a). Massha: an agent-based approach for human activity simulation in intelligent environments. *Pervasive and Mobile Computing*, 40:279–300.

Kamara-Esteban, O., PeÃśafiel, B., Pijoan, A., Borges, C., Berzosa, I., Alonso-Vicario, A., Badiola, J., and MartÃŋn, C. (2017b). Gis platform for the digitalization, data cleanup, and simulation of urban drainage networks. In *Proceedings of the 14th IWA/IAHR International Conference on Urban Drainage*, pages 2341–2344. IWA/IHR.

Kamara-Esteban, O., Pijoan, A., Alonso-Vicario, A., and Borges, C. E. (2017c). On-demand energy monitoring and response architecture in a ubiquitous world. *Personal and Ubiquitous Computing*, 21(3):537–551.

Kamara-Esteban, O., Sorrosal, G., Pijoan, A., Castillo-Calzadilla, T., Iriarte-Lopez, X., Macarulla-Arenaza, A. M., Martin, C., Alonso-Vicario, A., and Borges, C. E. (2016). Bridging the gap between real and simulated environments: a hybrid agent-based smart home simulator architecture for complex systems. In *Ubiquitous Intelligence & Computing, Advanced and Trusted Computing, Scalable Computing and Communications, Cloud and Big Data Computing, Internet of People, and Smart World Congress (UIC/ATC/ScalCom/CBDCom/IoP/SmartWorld), 2016 Intl IEEE Conferences*, pages 220–227. IEEE.

Karadimas, N. V., Rigopoulos, G., and Bardis, N. (2006). Coupling multiagent simulation and gis- an application in waste management. *WSEAS Transactions on Systems*, 5(10):2367–2371.

Karnouskos, S. and De Holanda, T. N. (2009). Simulation of a smart grid city with software agents. *EMS*, 9:424–429.

Keilson, J. (2012). *Markov chain models - rarity and exponentiality*, volume 28. Springer Science & Business Media.

Khattak, Z. H., Magalotti, M. J., Miller, J. S., and Fontaine, M. D. (2017). Using new mode choice model nesting structures to address emerging policy questions: A case study of the pittsburgh central business district. *Sustainability*, 9(11):2120.

Kiesel, J. and Wenkel, K.-O. (2007). Spatial generalization methods based on the moving window approach and their applications on landscape analysis. In Olgierd Hryniewicz, Jan Studzinski, M. R. E., editor, *Shaker Verlag*, pages 619–626. Shaker Verlag. ISBN 978-3-8322-6397-3.

Kormanyos, B. and Pataki, B. (2013). Multilevel simulation of daily activities: Why and how? In *2013 IEEE International Conference on Computational Intelligence*

and Virtual Environments for Measurement Systems and Applications (CIVEMSA), pages 1–6. IEEE.

Kothuri, R. K. V., Ravada, S., and Abugov, D. (2002). Quadtree and r-tree indexes in oracle spatial: a comparison using gis data. In *Proceedings of the 2002 ACM SIGMOD international conference on Management of data*, pages 546–557. ACM.

Krajzewicz, D., Erdmann, J., Behrisch, M., and Bieker, L. (2012). Recent development and applications of sumo-simulation of urban mobility. *International Journal On Advances in Systems and Measurements*, 5(3&4).

Krause, J. (2017). Introduction to pug. In *Programming Web Applications with Node, Express and Pug*, pages 81–87. Springer.

Kravari, K. and Bassiliades, N. (2015). A survey of agent platforms. *Journal of Artificial Societies and Social Simulation*, 18(1):11.

Kuhn, M. (2017). *caret: Classification and Regression Training*. R package version 6.0-78.

Kumar, M., Sarkar, P., and Madhu, E. (2013). Development of fuzzy logic based mode choice model considering various public transport policy options. *International Journal for Traffic and Transport Engineering*, 3(4):408–425.

Lai, L. L. (1998). *Intelligent system applications in power engineering: evolutionary programming and neural networks*. John Wiley & Sons, Inc.

Leach, P., Mealling, M., and Salz, R. (2013). Rfc 4122: A universally unique identifier (uuid) urn namespace, 2005. *Online: http://www. ietf. org/rfc/rfc4122. txt. Accessed*, 4.

Lemeshow, S., Sturdivant, R. X., and Hosmer, D. W. (2013). *Applied Logistic Regression*. Wiley.

Leprohon, M.-A. (2017). Ecmascript 6 and the evolution of javascript: A deeper look into the languageâĂŹs new features.

Lewis, D. D. (1998). Naive (bayes) at forty: The independence assumption in information retrieval. In *European conference on machine learning*, pages 4–15. Springer.

Lextrait, V. (2016 (accessed July 26, 2018)). The programming languages beacon.

Li, Y. and Muller, B. (2007). Residential location and the biophysical environment: exurban development agents in a heterogeneous landscape. *Environment and Planning B*, 34:279–295.

Lopez-Novoa, U., Mendiburu, A., and Miguel-Alonso, J. (2015). A survey of performance modeling and simulation techniques for accelerator-based computing. *IEEE Transactions on Parallel & Distributed Systems*, (1):272–281.

Loreto, S., Saint-Andre, P., Salsano, S., and Wilkins, G. (2011). Known issues and best practices for the use of long polling and streaming in bidirectional http. Technical report.

Luck, M., Noriega, P., Rodriguez-Aguilar, J. A., Sierra, C., et al. (2012). Communicating open systems. *Artificial Intelligence*, 186:38–94.

Luke, S., Cioffi-Revilla, C., Panait, L., and Sullivan, K. (2004). Mason: A new multi-agent simulation toolkit. In *Proceedings of the 2004 swarmfest workshop*, volume 8, pages 316–327. Michigan, USA.

Luke, S., Simon, R., Crooks, A., Wang, H., Wei, E., Freelan, D., Spagnuolo, C., Scarano, V., Cordasco, G., and Cioffi-Revilla, C. (2018). The MASON simulation toolkit: Past, present, and future. In *The 19th International Workshop on Multi-Agent-Based Simulation | MABS 2018*.

Lund, H. (2011). Energyplan-advanced energy systems analysis computer model. *Documentation version*, 9.

Ma, Y., Shen, Z., Zhou, D., and Wang, K. (2012). A planning tool for simulating urban growth process and spatial strategy of urban development in chuandong, china. In *Geospatial Techniques in Urban Planning*, pages 27–48. Springer.

Macal, C. M. and North, M. J. (2010). Tutorial on agent-based modelling and simulation. *Journal of simulation*, 4(3):151–162.

Marceau, D. J. and Benenson, I. (2011). *Advanced Geo-Simulation Models*. Bentham Science Publishers.

Martínez, F. L. and Morales, Y. O. (2012). Agent-based simulation approach to urban dynamics modeling. *Dyna*, 79(173):34–42.

Menassa, C. C., Kamat, V. R., Lee, S., Azar, E., Feng, C., and Anderson, K. (2013). Conceptual framework to optimize building energy consumption by coupling distributed energy simulation and occupancy models. *Journal of Computing in Civil Engineering*, 28(1):50–62.

Mendel, J. M. (2001). *Uncertain rule-based fuzzy logic systems: introduction and new directions*. Prentice Hall PTR Upper Saddle River.

Mendez-Vazquez, A., Helal, A., and Cook, D. (2009). Simulating events to generate synthetic data for pervasive spaces. In *Proceedings of the Workshop on Developing Shared Home Behaviour Datasets to Advance HCI and Ubiquitous Computing Research in Conjunction with CHI 2009*, pages 4–9, Boston, MA.

Meyer, J. and Downing, T. (1997). *Java virtual machine*. O'Reilly & Associates, Inc.

Moltchanov, B. and Rocha, O. R. (2014). A context broker to enable future iot applications and services. In *Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT), 2014 6th International Congress on*, pages 263–268. IEEE.

Müller, J. P. and Fischer, K. (2014). Application impact of multi-agent systems and technologies: A survey. In *Agent-oriented software engineering*, pages 27–53. Springer.

Nechaev, O. (2014 (accessed July 9, 2018)). Angular light.

Nikolai, C. and Madey, G. (2009). Tools of the trade: A survey of various agent based modeling platforms. *Journal of Artificial Societies and Social Simulation*, 12(2):2.

Nissen, S. (2005). Neural networks made simple. *Software*, 2(2):14–19.

Nissen, S. et al. (2006). Fann: fast artificial neural network library. *Avaliable at: http://leenissen. dk/fann/(02/12/2006)*.

North, M. J., Howe, T. R., Collier, N. T., and Vos, J. R. (2005). The repast simphony runtime system. In *Proceedings of the agent 2005 conference on generative social processes, models, and mechanisms*, volume 10, pages 13–15. Citeseer.

Okeyo, G., Chen, L., Wang, H., and Sterritt, R. (2012). Dynamic sensor data segmentation for real-time knowledge-driven activity recognition. *Pervasive and Mobile Computing*, 10:155–172.

O'Looney, J. (1997). *Beyond maps: GIS and decision making in local government*. ESRI, Inc.

Østergaard, P. A. (2015). Reviewing EnergyPLAN simulations and performance indicator applications in EnergyPLAN simulations. *Applied Energy*, 154:921–933.

Ouyang, M. (2014). Review on modeling and simulation of interdependent critical infrastructure systems. *Reliability Engineering & System Safety*, 121:43 – 60.

Overpass API (2016 (accessed July 9, 2018)). Overpass query language.

Pai, P.-F. and Hong, W.-C. (2005). Forecasting regional electricity load based on recurrent support vector machines with genetic algorithms. *Electric Power Systems Research*, 74(3):417–425.

Parsons, S., Rodriguez-Aguilar, J. A., and Klein, M. (2011). Auctions and bidding: A guide for computer scientists. *ACM Computing Surveys (CSUR)*, 43(2):10.

Passaglia, A. (2017). *Vue. js 2 Cookbook*. Packt Publishing Ltd.

Persen, T. and Winslow, R. (2016). Benchmarking InfluxDB vs MongoDB for time-series data, metrics and management. Technical report, Technical report, InfluxData, Inc., San Francisco, CA.

Piepho, H.-P. (2004). An algorithm for a letter-based representation of all-pairwise comparisons. *Journal of Computational and Graphical Statistics*, 13(2):456–466.

Pijoan, A. (2017). Geoworldsim website.

Pijoan, A., Borges, C. E., Oribe-Garcia, I., Martin, C., and Alonso-Vicario, A. (2015a). Agent based simulations for the estimation of sustainability indicators. *Procedia Computer Science*, 51:2943–2947.

Pijoan, A., Borges, C. E., Penya, Y. K., and Alonso-Vicario, A. (2014). Uso de algoritmos de enrutamiento para el cálculo de indicadores de sostenibilidad.

Pijoan, A., Esteban, O. K., Borges, C. E., and Penya, Y. K. Gis and mas tight coupling for spatial load forecasting.

Pijoan, A., Kamara-Esteban, O., Alonso-Vicario, A., and Borges, C. E. (2018). Transport choice modeling for the evaluation of new transport policies. *Sustainability*, 10(4):1230.

Pijoan, A., Kamara-Esteban, O., and Borges, C. E. (2015b). Environment modelling for spatial load forecasting. In *Agent Environments for Multi-Agent Systems IV*, pages 188–206. Springer.

Pijoan, A., Kamara-Esteban, O., Oribe-Garcia, I., Alonso-Vicario, A., and Borges, C. E. (2017a). Gtplat: Geosimulation for assessing the application of incentives to transport planning. In *Scientific And Technical Conference Transport Systems Theory And Practice*, pages 74–89. Springer.

Pijoan, A., Oribe-Garcia, I., Kamara-Esteban, O., Genikomsakis, K. N., Borges, C. E., and Alonso-Vicario, A. (2017b). Regression based emission models for vehicle contribution to climate change. In *Intelligent Transport Systems and Travel Behaviour*, pages 47–63. Springer.

Potter, M. A. and De Jong, K. A. (1994). A cooperative coevolutionary approach to function optimization. In *International Conference on Parallel Problem Solving from Nature*, pages 249–257. Springer.

Powar, A., Kamath, A., Gaikwad, K., and Dhruv, A. (2018). Survey on water resources information systems. *International Journal of Advanced Research in Computer Science*, 9(1).

Python user community (2017 (accessed July 9, 2018)). Phython's GIL controversy.

Qt Company (2014 (accessed June 9, 2018)a). Qt cross-platform application framework.

Qt Company (2018 (accessed July 9, 2018)c). Qvariant class.

Qt Company (2018 (accessed June 3, 2018)b). Qt framework, multithreading methodologies.

Quinlan, J. R. (1986). Induction of decision trees. *Machine learning*, 1(1):81–106.

R Core Team (2017). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria.

Rada-Vilela, J. (2017 (accessed June 9, 2018)). fuzzylite: a fuzzy logic control library.

Railsback, S. F., Lytinen, S. L., and Jackson, S. K. (2006). Agent-based simulation platforms: Review and development recommendations. *Simulation*, 82(9):609–623.

Ramsey, P. The state of open source gis.

Randall Munroe (2011). xkcd - A Webcomic - Significant. Accessed: 2015-12-13.

Rashidi, P. and Cook, D. (2011). Discovering activities to recognize and track in a smart environment. *Knowledge and Data Engineering, IEEE Transactions on*, 23(4):527–539.

Rendón Sallard, T. and Sànchez-Marrè, M. (2006). A review on multi-agent platforms and environmental decision support systems simulation tools.

Ricci, A., Rodriguez-Aguilar, J. A., Pijoan, A., and F., Z. (2015a). *Mixed environments for MAS: bringing the humans in the Loop*. Lecture Notes in Computer Science.

Ricci, A., Rodriguez-Aguilar, J. A., Pijoan, A., and Zambonelli, F. (2015b). Mixed environments for mas: Bringing humans in the loop. In *Agent Environments for Multi-Agent Systems IV*, pages 52–60. Springer.

Richardson, L. and Ruby, S. (2008). *RESTful web services*. " O'Reilly Media, Inc.".

Robinson, A. P., Duursma, R. A., and Marshall, J. D. (2005). A regression-based equivalence test for model validation: shifting the burden of proof. *Tree Physiology*, 25(7):903–913.

Rodrigues, D. S., Ribeiro, P. J. G., and da Silva Nogueira, I. C. (2015). Safety classification using gis in decision-making process to define priority road interventions. *Journal of transport geography*, 43:101–110.

Rodrıguez, J. (2001). On the design and construction of agent-mediated electronic institutions, vol. 14 of iiia monographs.

Roehr, T. (2014 (accessed July 26, 2018)). Fipa acl library.

Rojas, E. E. M., Castillo, J. N. P., and Torres, A. P. G. (2017). Visualization of urban flood areas of the amazon piedmont with multi-agents vectors natural geo-inspired (avng). *Facultad de Ingeniería*, 26(45):173–186.

Ronallo, J. (2012). Html5 microdata and schema.org. *Code4Lib Journal*, 16.

Rossman, L. A. et al. (2000). Epanet 2: users manual.

Rubinstein, R. Y. and Kroese, D. P. (2016). *Simulation and the Monte Carlo method*, volume 10. John Wiley & Sons.

Ruiz-Chavez, Z. and Salvador-Meneses, J. (2017). Simulation of evacuation with multi-agents on georeferenced layers with gama. *Latin American Journal of Computing Faculty of Systems Engineering Escuela Politécnica Nacional Quito-Ecuador*, 4(3):61–66.

Russell, E. and Wilensky, U. (2008). Consuming spatial data in netlogo using the gis extension. In *The annual meeting of the Swarm Development Group*.

Russell, S. J., Norvig, P., Canny, J. F., Malik, J. M., and Edwards, D. D. (2003). *Artificial Intelligence: A Modern Approach*. Prentice Hall.

Satyanarayanan, M. (2017). The emergence of edge computing. *Computer*, 50(1):30–39.

Schalkoff, R. J. (1997). *Artificial neural networks*, volume 1. McGraw-Hill New York.

Schema.org Community (2018b). Person in schema.org. Online (accessed June 3, 2018).

Schema.org Community (2018 (accessed June 3, 2018)a). Organisation in schema.org.

Schema.org Community (2018 (accessed June 3, 2018)c). Place in schema.org.

Sengupta, R. and Sieber, R. (2007). Geospatial agents, agents everywhere... *Transactions in GIS*, 11(4):483–506.

Senjyu, T., Takara, H., Uezato, K., and Funabashi, T. (2002). One-hour-ahead load forecasting using neural network. *Power Systems, IEEE Transactions on*, 17(1):113–118.

Shaw, S.-L. and Xin, X. (2003). Integrated land use and transportation interaction: a temporal gis exploratory data analysis approach. *Journal of transport geography*, 11(2):103–115.

Sierpiński, G., Celiński, I., and Staniek, M. (2014). Using trip planners in developing proper transportation behavior. *International Science Index*, 8(11):482–490.

Sierpiński, G., Staniek, M., and Celiński, I. (2016). Travel behavior profiling using a trip planner. *Transportation Research Procedia*, 14:1743–1752.

Siskos, P. and Capros, P. (2014). Primes-tremove: a transport sector model for long-term energy-economy-environment planning for eu. In *20th Conference of the International Federation of Operational Research Societies in Barcelona*.

Skinner, I. (2014). The Mitigation of Transport's CO_2 Emissions in the EU: Policy Successes and Challenges. *Forthcoming in Van Calster, G., Vandenberghe, W., and Reins, L.(eds), Research Handbook on Climate Mitigation Law, Cheltenham, Edward Elgar*.

Souche, S. (2010). Measuring the structural determinants of urban travel demand. *Transport policy*, 17(3):127–134.

Souza, A. M. and Amazonas, J. R. (2013). A novel smart home application using an internet of things middleware. In *Smart Objects, Systems and Technologies (SmartSysTech), Proceedings of 2013 European Conference on*, pages 1–7. VDE.

Staniek, M. and Sierpiński, G. (2017). Cost criteria as a means to support travelling mode related decisions âĂŤ a case study for the central part of silesian voivodeship (poland). *Sustainable Transport Development, Innovation and Technology*, pages 137–150.

Steiner, R., Leask, G., and Mili, R. (2006). An architecture for mas simulation environments. In *Environments for Multi-Agent Systems II*, pages 50–67.

Sujil, A., Verma, J., and Kumar, R. (2018). Multi agent system: concepts, platforms and applications in power systems. *Artificial Intelligence Review*, 49(2):153–182.

Sullivan, K., Coletti, M., and Luke, S. (2010). Geomason: Geospatial support for mason. Technical report, Department of Computer Science, George Mason University.

Sutter, H. (1999). Optimizations that aren't (in a multithreaded world). *C/C++ Users Journal*, 17(6):61–69.

Synnott, J., Nugent, C., and Jeffers, P. (2015). Simulation of Smart Home Activity Datasets. *Sensors (Basel, Switzerland)*, 15(6):14162–79.

Taillandier, P., Vo, D.-A., Amouroux, E., and Drogoul, A. (2010). Gama: a simulation platform that integrates geographical information data, agent-based modeling and multi-scale control. In *International Conference on Principles and Practice of Multi-Agent Systems*, pages 242–258. Springer.

Tapia, E. M., Intille, S. S., and Larson, K. (2004). *Pervasive Computing*, volume 3001 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, Berlin, Heidelberg.

Teodosiu, C., Ardeleanu, C., and Lupu, L. (2009). An overview of decision support systems for integrated water resources management. *Environmental Engineering & Management Journal (EEMJ)*, 8(1).

Terano, T. (2008). Beyond the kiss principle for agent-based social simulation. *Journal of Socio-informatics*, 1(1):175–187.

Tilkov, S. and Vinoski, S. (2010). Node. js: Using javascript to build high-performance network programs. *IEEE Internet Computing*, 14(6):80–83.

Tisue, S. and Wilensky, U. (2004). Netlogo: A simple environment for modeling complexity. In *International conference on complex systems*, volume 21, pages 16–21. Boston, MA.

Torrens, P. M. (2003). Cellular automata and multi-agent systems as planning support tools. In *Planning support systems in practice*, pages 205–222. Springer.

Torrens, P. M. (2010). Agent-based models and the spatial sciences. *Geography Compass*, 4(5):428–448.

Tuzkaya, U. R. and Önüt, S. (2008). A fuzzy analytic network process based approach to transportation-mode selection between turkey and germany: A case study. *Information Sciences*, 178(15):3133–3146.

Ugander, J., Karrer, B., Backstrom, L., and Marlow, C. (2011). The anatomy of the facebook social graph. *arXiv preprint arXiv:1111.4503*.

Valckenaers, P., Sauter, J., Sierra, C., and Rodriguez-Aguilar, J. A. (2007). Applications and environments for multi-agent systems. *Autonomous Agents and Multi-Agent Systems*, 14(1):61–85.

Vapnik, V. N. and Chervonenkis, A. Y. (1971). On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability & Its Applications*, 16(2):264–280.

Velleman, J. D. and Bratman, M. E. (1991). Intention, plans, and practical reason. *The Philosophical Review*, 100(2):277.

Waddell, P. (2002). Urbansim: Modeling urban development for land use, transportation, and environmental planning. *Journal of the American planning association*, 68(3):297–314.

Walker, E. and Nowacki, A. (2011). Understanding equivalence and noninferiority testing. *Journal of General Internal Medicine*, 26(2):192–196.

Weyns, D. (2014). E4MAS - environments for multiagent systems - 10 years later.

Weyns, D., Omicini, A., and Odell, J. (2007a). Environment as a first class abstraction in multiagent systems. *Autonomous agents and multi-agent systems*, 14(1):5–30.

Weyns, D., Omicini, A., and Odell, J. (2007b). Environment as a first class abstraction in multiagent systems. *Autonomous agents and multi-agent systems*, 14(1):5–30.

Weyns, D., Parunak, H. V. D., Michel, F., Holvoet, T., and Ferber, J. (2004). Environments for multi-agent systems. In *First International Workshop (E4MAS 2004)*, volume 3374.

Weyns, D., Parunak, H. V. D., Michel, F., Holvoet, T., and Ferber, J. (2005). Environments for multiagent systems state-of-the-art and research challenges. In *Environments for multi-agent systems*, pages 1–47. Springer.

White, R. (1998). Cities and cellular automata. *Discrete dynamics in Nature and Society*, 2(2):111–125.

Willis, H. (1996). *Spatial Load Forecasting*. Marcel Dekker Inc., New York.

Wooldridge, M. and Jennings, N. R. (1995). *Intelligent agents: Theory and practice*, volume 10. Cambridge University Press.

Xia, F., Yang, L. T., Wang, L., and Vinel, A. (2012). Internet of things. *International Journal of Communication Systems*, 25(9):1101–1102.

Yang, R., Lau, W. C., and Liu, T. (2016). Signing into one billion mobile app accounts effortlessly with oauth2. 0. *blackhat Europe*.

Yue, C. and Wang, H. (2009). Characterizing insecure javascript practices on the web. In *Proceedings of the 18th international conference on World wide web*, pages 961–970. ACM.

APPENDIX

A

JavaScript Asynchrony

Many of the GeoWorldSim modules allow functionality extension by dynamically loading and asyncrhonously execution pieces of code in JavaScript. In order to achieve such, it is important to understand how this programming language's internals. JavaScript is an event-driven and non-blocking I/O programming language which uses a single thread per process. This is a very deliberate design decision that eliminates the need to deal with locking semantics. Asynchronous functions, request external software to perform some computation (e.g. a database to fetch records, an web server to response, file system to read a file) and decouple the control from the execution flow of Javascript. It is done so, to avoid heavy processes blocking the single Javascript event loop waiting for the response of the external software. In an asynchronous function, it is not possible to execute the *return* statement, since values are not ready in time. This asynchronous world control has led programmers to find different approaches for managing how to return the result back to the execution flow.

Because functions are first-class objects, it is possible to pass a function as an argument for another function and later execute that passed-in function. Passing a callback function has been for several years a widely used mechanism in NodeJS for returning asynchronous results following the Continuation Passing paradigm (Appel and Jim, 1989). It consists of passing the control explicitly as an argument in the form of a continuation, that is, a callback function. As the name says, callback functions are created for an activity to take place whenever a previous event completes. Therefore, a callback is passed to the the external software so when finishing the asynchronous computation it can invoke the callback and send the result in it. Listing A.14 shows a small example of an external asyncrhnous database call which needs to be passed a callback function to return the query result. Callbacks are ordinary functions that can receive several arguments. In NodeJS the *error-first* technique has been for several years adopted as a standard for managing callbacks establishing that callbacks will return in the first place the error (if happened) and then the rest of results.

Having to pass a callback at the end of each function, derived in the known as *callback hell*. It happens when programmers start nesting functions trying to write JavaScript in a way where execution happens visually from top to bottom, the code takes a pyramid shape and it becomes difficult to follow and debug. In 2015, the ECMAScript 6 Standard (Leprohon, 2017) added support for the concept of Promises, a first class representation of a value that may be made asynchronously and be available in the future. It became popular after several Javascript libraries such as Dojo, Q or Jquery had created their own similar method for receiving deferred results. Promises finally suppose a native mechanism for managing asynchrony and enable programmers to chain asynchronous computations as if they were sequential providing a direct correspondence between synchronous functions and asynchronous functions.

Promise users can chain two methods to handle the fulfilled value or the reason for rejection, acting as a black box. The code looks sequential by chaining promises through the `then()`, if the promise had no error, and `catch()`, if there was an error, statements. For this to work, Promises are a wrapper for asynchronous code that always receive two `resolve()` and `reject()` methods. These methods provide

transparent result and error returning in the asynchronous world. Programmers can chose between `resolve()` (to return the asynchronously obtained value) or `reject()` (to notify the occurred error) and pass a single value to them, which will be passed to the functions chained with `then()` and `catch()`. Chaining promises, fulfilment and rejection will compose just like their synchronous counterparts, with fulfilment flowing up a composition chain and being interrupted at any time by a rejection. Listing A.15 shows the implementation of an asynchronous process by chaining promises.

Inside GeoWorldSim's ecosystem, many modules wrap the extensible functionality in Promises as interface for other users to extend. This is the case of:

Datasources module connectors: Code for reading data (e.g. database drivers, file readers) are put inside Promises and extracted out of the main software into external files for users to be able to add more new connectors.

Logic building module blocks: Any logic block is wrapped inside a Promise. Then all these are executed in sequence and passing the results from one to the other in order to create complex logic flows. All blocks are external files injected into the sequence for users to be able to add more new of them.

Alerts module forwarders: When an alert arrives, it is possible to forward it to another software (e.g. email account, SMS sender, endpoint). To this end, Alerts module's forwarders are Promise based external files for users to create more ways of forwarding or processing an alert.

Promises are a great mechanism for better asynchronous coding but also a far cry from the DRY[1] principle. When using a Promise a big piece of code needs to be repeated to properly manage the application's flow. Lastly, within the ECMAScript 8 standard (Ecma International, 2018), new `Async` and `Await` statements have been added to JavaScript for better reading and writing of code. In spite of using Promises underneath, `async/await` keywords allow writing Promise-based code as if it were synchronous (as shown in listing A.16), but without blocking the main thread. This

[1] Don't Repeat Yourself

new advance, has become the best tool for asynchronous and elegant JavaScript programming.

```
function getRecords(){
    database.query( "SELECT * FROM clients" , printRecords );
    // The database engine will call printRecords
    // once it has the results passing firstly if
    // there was an error and secondly the fetched records.
}

function printRecords( error , records ){
    if( error ){
        console.error( 'Error when querying : ' + err );
    } else if( !records ){
        console.error( 'Empty records' );
    } else {
        filseystem.writeFile( records , finished );
    }
}

function finished( err ){
    if( error ){
        console.error( 'Error when printing : ' + err );
    } else {
        console.log( 'All finished!' );
    }
}
```

Code A.14: NodeJS function calling an external database engine and passing a callback. Once launching the query, NodeJs decouples the execution flow and continues serving other requests. When the database engine finishes fetching the results, it invokes `printRecords()` callback function with the returning arguments and NodeJS will serve it as another emerged request. The printing function then calls the filesystem to asynchronously print a file and when it finished to call a `finished()` callback to check if there were errors. (Source: own research)

```
database.query( "SELECT * FROM clients" )
.then( records => {
    if( !records ){
        throw 'Empty records';
    }
    return filseystem.writeFile( records );
})
.then( () => {
    console.log( 'All finished!' );
})
.catch( err => {
    console.error( 'Error when querying or printing : ' + err );
});
```

Code A.15: The same above example but implemented using Promises. These can be chained using the `then()` method and the interruption can be attended at any moment of the flow. (Source: own research)

```
const records = await database.query( "SELECT * FROM clients" );
if( !records ){
    throw 'Empty records';
}
await filseystem.writeFile( records );
console.log( 'All finished!' );
```

Code A.16: The same above example but implemented using Async/Await. (Source: own research)

A.1 Lazy loading

Above has been presented how the asynchronous code should look for being processed by the server but it is still pending to know how to read this code from a file. JavaScript contains several mechanisms for defining modules such that the module and its dependencies can be asynchronously loaded. This is particularly well suited for the browser environment where synchronous loading of modules incurs performance, usability, debugging, and cross-domain access problems. The most basic form is using the native `eval()` function which takes a string and runs it as if it was plain JavaScript code. This functionality is as powerful as dangerous since it provide access to the JavaScript compiler. It is sometimes necessary, but in most cases it indicates the presence of extremely bad coding (Yue and Wang, 2009) given that:

Debugging: It makes harder to work with a debugger because it is like running code into a black box and then out of it. Even despite knowing with certainty that the *stringed code* works, using `eval()` requires extra effort to debug the real code together with the one generated by `eval()`. Even more, `eval()` disables the possibility of using code minimisers or compressors. These tend to shorten variable names that are in scope of the call to `eval()` but do not change them inside the *stringed code*. Since `eval()` can access any of those variables directly, renaming them would introduce errors.

Performance: The script compiler cannot pre-compile code in `eval()` as it does not know what the code will contain until it gets there. Many JavaScript engines can run code in one of two ways: fast path or slow path. Fast path refers to code that is stable and predictable, and can therefore be compiled for faster execution. Slow path code is unpredictable, making it hard to compile and may still be run with an interpreter. The mere presence of `eval()` in the code makes it unpredictable and therefore will run in the interpreter, making it run at *old browser* speed instead of *modern browser* speed (once again, a 10x difference).

Security: Hackers love this function because it is always easier to inject a string into a program than to inject code. Cross-Site Scripting (XSS) is a type of computer

security vulnerability typically found in web applications. It enables, on the one hand, attackers to inject client-side scripts into web pages viewed by other users and, on the other, users to alter the functioning of a web by injecting code. The first, refers to downloading a script that needs to be `eval()`ed to perform some functionality. Browsers usually make cross-domain checks to ensure that downloaded and injected code comes from trustful sources. However having a *man-in-the-middle* attack could send and make any type of code run in a user's browser. The second vulnerability, is the same case as with the well-known SQL injection: if users know the underlying `eval()` function that is being executed (just like knowing the underlying SQL statement), they can enter a code that uses the function to perform a different operation from what it was supposed to do.

Speaking more specifically about JavaScript for servers, with NodeJS contains a `require()` built-in function for lazy-loading of all its native and external requirements. Modules are the fundamental building blocks of the code structure and NodeJS implements a core system called `module.js` to manage these which controls the loading, compiling, and caching of every file imported. The `require()` function is the method for lazy-loading a file from a string literal. The Module System handles:

Finding Resolves the absolute path of the required file by checking in the project directory, installed modules and paths specified by module.paths.

Loading It determines the type of content of the file to load whether it is JavaScript, JSON data or even a C++ file.

Scoping Wrapping the file in its private scope by hiding information and only exposing the public interface of the code using the `module.exports` structure. In any module, `exports` is a special object to store functions and properties that will be public to the code requiring this module.

Evaluating Compile the code and populate the exports of the module with the loaded functionality.

Caching the module so that when it is required again it does not need to repeat all the steps. If circular dependency between modules occurs (e.g. module1 requires module2 and vice-verse) NodeJS manages it simple. During the loading of a module, it builds the exports empty object which will gradually be completed. Code requiring the module before it has done loading and will receive a partial (but existing) exports object with whatever was defined so far.

Therefore, behind the scenes, `require()` function makes an evaluation of code or data inside a file loads it just the same way as `eval()` would do. However it brings several important additions like easy file loading, scoping the evaluated code into a module block or caching. Wrapping the code ensures it can only access its resources and exposes only desired data for others to use. Solves in a way some of the problems of the `eval()` function while providing access to all the native and third party modules of the NodeJS ecosystem.

Combining lazy-loading of code into modules plus the power of asynchronous functions, GeoWorldSim tools make possible their extension and adding of new functionality. Extensions only need to be programmed as an *async function* that creates and returns *Promise* inside. The *Promise* will perform the desired working and then results and errors are returned through the `resolve()` and `reject()` methods as described in listing A.17.

```
// Code inside the server
const data_to_process = [...];
const processor_file = 'my-custom-processor.js';

// Load the custom implementation from a file and execute with await
const data_processed = await require( processor_file )( data_to_process );
if( !data_processed ){ 'Error occurred'; }

// Load the custom implementation from a file and execute with promises
require( processor_file )( data_to_process )
.then( data_processed => {
    [...];
})
.catch( err => {
    console.log( 'Error ocurred' , err );
});
```

```
// Code inside my-custom-processor.js
module.exports = async function( data_to_process ) {
    return new Promise( ( resolve , reject ) => {
        try {

            // Implement the processing procedure

            // If success return the result using the resolve callback
            let data_processed = [];
            resolve( data_processed );

        } catch ( err ){

            // If error return the message using the reject callback
            reject( err );
        }
    });
}
```

Code A.17: Combination of asynchronous functions with lazy-loading of extensions. (Source: own research)

APPENDIX

B

Web views mixed rendering methodology

Using a website by us humans, requires transforming the information into readable content, lists and illustrations. For this purpose the web server generates the required HTML and the necessary programming for a user to view a web page. This is typically achieved using server-side rendering, that is, when a web browser requests data, the server gathers it, writes the entire HTML code and returns the rendered webpage. With this, web browsers need to only remove the previous page and straight display the received rendered page. When server-side rendering, there is no distinction whether the new page only has a few items that are different than the current page, the browser will ask for the entire new page and will re-render everything from the ground up. These implementations transmit significant amounts of data to the client device, and force the browser to create Document Object Model (DOM) (Document Object Model) objects for every markup element generated by the server. As these DOM trees get larger web browser performance deteriorates.

Client-side rendering, refers to the technique of rendering content in the browser using JavaScript. Instead of getting all of the content rendered in the HTML document itself, the server sends a bare-bones HTML document with a JavaScript file that will afterwards alter the document in the browser to inject the rest of the site. It is not a new approach () but it did not become really popular until JavaScript libraries started incorporating it into their style of development. Some notable examples of these are Angular.js (Jain et al., 2015), Vue.js (Passaglia, 2017) and React.js (Fedosejev, 2015).

All the web views of the GeoWorldSim platform use a mix of server and client side rendering methodology for a better user experience. Combining both contributes to a better user experience since every page only requires loading a very small section of the page to display the new content, instead of loading the entire page at every interaction. To this end every time a user accesses any module, the browser firstly downloads an empty HTML skeleton further detailed in Appendix B.0.1 with some basic structure and the Javascript logic for rendering the page. This combined server and client side rendering brings advantages in:

Reduce server load: Servers only need to render a little amount of HTML views speeding the initial load. The computation power needed to process and render data is gathered from the user devices instead of the server. This is very related with the Edge computing method (Satyanarayanan, 2017) which tries to optimise resources by taking computation out of the central nodes to the edges where data is produced or requested.

Faster navigation: During the initial web load, users rapidly receive the root page where then all the internal sections go asynchronously loading. Afterwards, users also feel a more fluent navigation as the majority of the page remains still and only few sections change when interacting with them.

Lazy loading: Client-side rendering supports postponing section and elements loading until when the user needs to visualise them (e.g. on scrolling down, on searching) reducing bandwidth. If some information it is not needed, there server will not be queried about.

Reach interaction: Client-side rendering supports animated interactions and transitions. In server side rendering each navigation loads the entire page again and

therefore Javascript initialises all variables again losing prior state not being able to animate transitions. Despite not being a critical feature, eye catching animations are always a plus when it comes to making an attractive product.

I took the decision of limiting to 5 the amount of views that would be rendered at the server side, which are the generic views all modules have in common (the traditional CRUD operations). Additional module specific web views must be done in the client accessing the data through the REST API. For server side HTML is compilation GeoWorldSim modules use Pug and Django templates. Pug (Krause, 2017) is a high-performance template engine implemented with JavaScript for NodeJs. HTML involves major code repetition since every element is named twice: once to delimit the start of its content and once to delimit the end. Pug avoids all of this by relying on indentation to determine where elements and blocks of code begin and end. Not only does this result in smaller code writing, it makes the code much cleaner to look at. Pug and Django templates both allow splitting templates into separate files and contain instructions to dynamically generate content and maximise code reuse. Imagine a complex HTML form which needs to appear in several pages, these template engines permit writing this code once in a separate single file and then importing it in all the pages where it is needed. Listing B.18 shows an example pug file which results in the compiled Listing B.19. So, this technology renders at the server the following views:

- Root index page with the HTML skeleton where to inject all the content.
- List view to depending on the module display the data-sets, static files, simulations, etc of a user. Sharing the same view ensures that list style and interfaces are homogeneous among modules.
- Creation / updating view to set up a new element (e.g static file, datasource or simulation). Using the template structure each module contains its customized view.
- Read view to see the status and attributes of an element.
- Delete view to remove one element from the module.

```
html(lang="en")
  head
    title Pug
  body
    h1 Pug - node template engine
    #container.bordered
        p.hoverFlip.underlined You are amazing
```

Code B.18: Example HTML page written in pug template engine. (Source: own research)

```
<html lang="en">
  <head>
    <title>Pug</title>
  </head>
  <body>
    <h1>Pug - node template engine</h1>
    <div id="container" class="bordered">
      <p class="hoverFlip underlined">You are amazing</p>
    </div>
  </body>
</html>
```

Code B.19: Result of the pug compiled page. (Source: own research)

B.0.1 HTML skeleton

All web sites usually contain an index page (located at the root / of the page) from which to navigate through other sections. The index page of GeoWorldSim modules, is built as a skeleton composed of three sections where information is injected from several modules and sources:

Header: Rendered at the top, it contains the title, description and logo image of the page together with navigation menus, or other navigation functionality. All headers, regardless of which module they belong, are loaded from the Auth module. This module is who identifies users and what services they have access to, therefore it is logical to retrieve the rendered navigation menus and links to resources from the Auth module. Initially, if the user has yet not been

identified, the Auth module will return an anonymous navigation bar with with no user information nor menus. When the user identifies, the module sends the information to the Auth server and this returns a user identification token which will be used for requesting the navigation bar in all subsequent navigation. Using this token, the personalised navigation bar will be obtained from the Auth module for injecting it in the HTML skeleton header.

Main content: Section where the content changes at every user navigation or element transformation. It consists of several server-side rendered views are injected at this point and then modified by the client side logic. The code at the web browser makes use of Angular Light framework, more explained bellow.

Footer: Footer is rendered by the server of each module at the bottom of the skeleton, containing copyright and attribution notices.

In order to establish a unique user experience and style across all modules, their websites must comply with the Material Design principles (). Material is a design normative developed in 2014 by Google intended to unify interfaces across platforms, devices, and input methods. It uses card components and z-axis elevation imitation to design all the components and usage in an intuitive user experience. To satisfying the Material Design philosophy GeoWorldSim makes use of MaterializeCSS, a library combining HTML, CSS, and JavaScript code designed to help build user interface components. It contains well defined CSS styles providing out-of-the-box principles for good design and elegant styling.

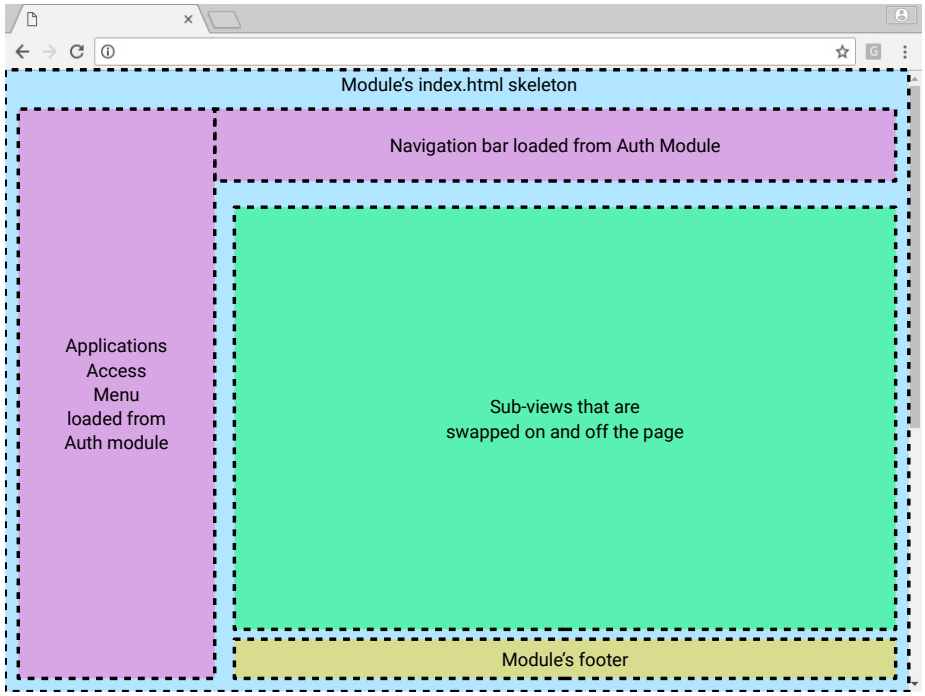


Figure B.1: Layout of the web views that constitute the user interface. (Source: own research)

B.0.2 **Angular Light**

Angular.js is one of the most demanded JavaScript front-end structural framework for dynamic web apps. It takes many server-side typical operations to the browser and allows using HTML as template language extending the syntax into a more expressive and readable format. Angular's data binding and dependency injection eliminates much of the code that is commonly repeated in every web application. It uses a construct called directive which allows to decorate HTML with special markup that synchronises with JavaScript variables enabling not only describing content in the HTML but also logic. However, its data binding model and extent list of tools makes it big in size and can get the graphical interface to run severely slow. In search for

a lighter solution, Oleg Nechaev developed in 2014 the Angular Light framework (Alight) (Nechaev, 2018) based on several web frameworks and similar to a simplified Angular version. This is the key technology that shapes the logic in the web browser.

Alight extends the declarative language in HTML documents teaching the browser new syntax through a construct of directives. Examples include:

- Data binding, using {{}}.
- DOM control structures for repeating, showing and hiding DOM fragments.
- Attaching new behaviour to DOM elements, such as DOM event handling.
- Support for forms and input validation.

At a high level, directives are markers on a DOM element (such as an attribute, element name, comment or CSS class) that tell Alight's HTML compiler to attach a certain behaviour to that element (e.g. via event listeners) or even to transform the element and its children. Whenever the web browser finds a known directive, it will execute the associated behaviour. Alight comes with several built-in directives such as: `al-model`, two-way binding to user input elements; `al-repeat`, for iterating an element and rendering all its elements; `al-if`, to render this element if the given condition is fulfilled; and `al-html`, to include a block of HTML. User created directives can contain variables and functions which be available within the entire block they are attached to, that is to its DOM element and all the children. For every directive, a `scope` structure is created in which every variable and function of the scope is double bind in the Javascript code and DOM. Changes in the value at Javascript level will be propagated to the HTML view and vice verse. Scopes of directives are *inherited* to child element directives, that is if a parent DOM element has a directive attached to it, a directive attached to a child DOM element will have its parents data inside its scope. This methodology ensures all functionality is encapsulated to be only available in the code it is needed.

B.0.3 **Directives**

GeoWorldSim web views make extensive use of Alights built-in and custom developed directives for client side rendering and data manipulation. Every directive has a

`scope` object in which to insert the double binded data, an `elm` object containing the DOM element it has been attached to and a `env` object with Alight's functions to interact with the HTML (e.g. `env.takaAttr()` provides access to attributes of the DOM element, `env.apply()` spreads the changes made in the code to the HTML view). Listing B.20 presents the HTML usage of two directives.

`al-entities`: Adding this directive results in automatically executing a request to fetch the entities from the server using its REST API. The directive requires the name of the entities to fetch and the pagination. It will load in the scope structure both the downloaded entities and the total count returned by the API.

```
alight.directives.al.entities = function(scope, elm, exp, env) {
    scope.page_size = env.takeAttr( 'page-size' ) || 9;
    scope.page = env.takeAttr( 'page' ) || 0;
    scope.entity_name_plural = env.takeAttr( 'entity-name-plural' );

    $.ajax({
        url : '/api/' + scope.entity_name_plural + '/offset/'
                      + scope.page * scope.page_size + '/limit/'
                      + scope.page_size ,
        type : 'GET'
    })
    .then( res => {
        scope.count = res.count;
        scope[scope.entity_name_plural] = res[scope.entity_name_plural];
        env.scan();
    })
    .catch( err => { console.log });

};
```

`al-entity`: It will automatically request an entity to the REST API. It needs the entity type and id to make the request.

`al-entityops`: The HTML element and children who get attached this directive will be able to invoke entity create, update and delete API calls. Creating and updating require the entire entity to be passed while for deleting only entity type and ID are needed. It allows sending an extra parameter for redirection after the operation has succeeded.

```
alight.directives.al.entity = function( scope, elm, exp, env ) {

    scope.id = env.takeAttr('id');
    scope.entity_name_singular = env.takeAttr( 'entity-name-singular' );

    $.ajax({
        url : '/api/' + scope.entity_name_singular + '/' + scope.id ,
        type : 'GET'
    })
    .then( res => {
        scope[scope.entity_name_singular] = res;
        env.scan();
    })
    .catch( err => { console.log  });

};
```

al-login: Makes login, logout and isloggedin functions available for the block of code attached. For any of these operations a success-redirect and fail-redirect paths can be defined to customise redirection after execution.

al-fileread: Adds an event listener for changes in a file input field. When selecting a file the directive will create a FileReader to get the local file, parse it and store it in the scope.

al-map: Automatically create a Leaflet (Agafonkin) interactive map in the attached DOM element. The map is loaded with the Wikimedia, OpenStreetMap, Toner and Satelite layers and inserted in the scope for later manipulation. Additionally the directive exposes functions to draw elements in the map (e.g. draw an agent, draw a point, draw a polygon) and retrieve information from it (e.g. get map bounds, get map zoom level).

al-websocket: Contains all the logic to open a WebSocket listenning the communication bus of a simulation. It requires the simulation ID and being child of a `al-map` directive as it will call the map functions to render the agents. Additionally, displays socket messages using Materialize's Toasts.

al-jsontable: Reads a complex JSON object from the scope and renders it as a table (`<table>`) in the attached DOM element. The directive iterates all the keys of the JSON object and creates a table row (`<tr>`) with the key at the first column (`<th>`) and the value at the second (`<td>`). If this value is

another object or array inside the cell a new table is be created and the value is recursively iterated repeating the above process.

```
alight.directives.al.entityops = function( scope, elm, exp, env ) {

    scope.entity_name_singular = env.takeAttr( 'entity-name-singular' );
    scope.create = function( data ){
        // Create entity
        $.ajax({
            url : '/api/' + scope.entity_name_singular ,
            type : 'POST',
            data : JSON.stringify( data ),
            dataType: "json"
        })
        .then( res => {
            location = env.takeAttr( 'then-redirect' ) || '/view/'
            + scope.entity_name_singular + '/view?id=' + res.id ;
        })
        .catch( err => { console.log });
    };

    scope.update = function( id , data ){
        // Update entity
        $.ajax({
            url : '/api/' + scope.entity_name_singular + '/' + id ,
            type : 'PUT',
            data : JSON.stringify( data ),
            dataType: "json"
        })
        .then( res => {
            location = env.takeAttr( 'then-redirect' ) || '/' ;
        })
        .catch( err => { console.log });
    };

    scope.delete = function( id ){

        if( confirm("Are you sure?") ){
            // Delete entity
            $.ajax({
                url : '/api/' + scope.entity_name_singular + '/' + id,
                type : 'DELETE'
            })
            .then( res => {
                location = env.takeAttr( 'then-redirect' ) || '/' ;
            })
            .catch( err => { console.log });
        };

    };
};
```

```
alight.directives.al.loginbuttons = function(scope, elm, exp, env) {

    scope.login = function( data ){
         $.ajax({
            url : '/api/login' ,
            type : 'POST',
            data : data
        })
        .then( res => {
            location = env.takeAttr( 'then-redirect' ) || '/' ;
        })
        .catch( err => { console.log( err ); });
     };

    scope.logout = function(){
        $.ajax({
            url : '/api/logout/' ,
            type : 'GET'
        })
        .then( res => { location = '/'; })
        .catch( err => {  console.log( err ); });
    };
};

alight.directives.al.isloggedin = function( scope, elm, exp, env) {

    $.ajax({
        url : 'http://auth.deusto.io/api/me',
        type : 'GET'
    })
    .then( res => {
        console.log( res );
        location = env.takeAttr( 'then-redirect' ) || '/';
    })
    .catch( err => {
        console.log( err );
        location = env.takeAttr( 'catch-redirect' ) || '/view/login';
    });
}
```

```
alight.directives.al.fileread = function(scope, elm, exp, env) {
    var store_var = exp || 'file';
    elm.addEventListener("change" , function(evt) {

        var file = evt.currentTarget.files[0];
        var reader = new FileReader();
        reader.onload = function(evt) {

            var json_read = JSON.parse( evt.target.result );
            if( json_read ){
                scope[ store_var ] = json_read;
                env.scan();
            } else {
                alert( 'Could not parse file' );
            }
        };
        reader.readAsText( file );
    });
};
```

```
alight.directives.al.map = function(scope, elm, exp, env) {

    var map_name = env.takeAttr( 'map-name' ) || 'map';
    var map_container_id = exp;

    scope.overlays = scope.overlays || {};
    scope[ map_name ] = L.map( map_container_id ,
    { fullscreenControl : true , zoomSnap : 0 })
    .setView([46.058, 14.304], 4);

    var baselayers = {...};
    // Add scale
    L.control.scale().addTo( scope[ map_name ] );

    // Layer switcher
    scope.layerSwitcher =
    L.control
    .layers( baselayers , scope.overlays , { position : 'bottomleft' })
    .addTo( scope[ map_name ] );
    ctrl.addTo( scope[ map_name ] );
};
```

```
alight.directives.al.mapEntities = function(scope, elm, exp, env) {

    scope.overlays = scope.overlays || {};
    var layers = {};
    var map_name = env.takeAttr( 'map-name' ) || 'map';

    scope.mapEntity = function( entity ) {
        var id = entity.id;
        var type = entity.type;

        // If new entity class
        if( !layers[type] ){
            layers[ type ] = {};
            scope.overlays[ type ] = L.featureGroup();
            scope.overlays[ type ].addTo( scope[ map_name ] )
            scope.layerSwitcher.addOverlay( scope.overlays[ type ] , type );
        }

        // Delete if existing layer
        if( layers[type][id] ){
            scope.overlays[ type ].removeLayer( layers[ type ][ id ] );
            delete layers[ type ][ id ];
        }

        // Create geojson layer and add it to scope.overlays
        if( entity.geometry ){
            var geojson =
            { "type": "Feature" , "properties": {} , "geometry" : entity.geometry.value };
            for(var p in entity){ // Set properties
                if( p != 'geometry' ){ geojson.properties[p] = entity[p]; }
            }

            var layer = L.geoJson( geojson ,
            { style : st , pointToLayer : ptl , onEachFeature : oef } );
            layers[ type ][ id ] = layer;
            scope.overlays[ type ].addLayer( layer );
        }
    };

    var st = function( f ){
        var style = f.properties.style || {};
        return {
            fillColor : style.color ? style.color.value : '#666',
            fillOpacity : 0.2,
            color : style.border_color ? style.border_color.value : '#000',
            weight : style.weight ? style.weight.value : 3,
            iconSize : [46, 46],
            radius: 5,
        }
    }

    var ptl = function(feature, latlng) {
        //return L.circleMarker( latlng , { style : st , onEachFeature : oef } );
        return L.marker( latlng , { icon : L.icon( getStyle(feature) ) });
    }
}
```

```
alight.directives.al.websocketListener = {
    link : function(scope, elm, exp, env) {
        var id = exp;
        id = env.takeAttr('id');
        var waited = 5;

        var connectSocket = function(){

            $.get('http://sockets.geoworldsim.com/api/socket/' + socket_id)
            .then( res => {

                if( !res || !res.socket ){ throw 'Socket not open' }

                M.toast({ html : 'Simulation is active' });

                var socket = new WebSocket('ws://sockets.deusto.io');
                socket.onopen = function () {
                    socket.send( JSON.stringify( { signal : 'join' ,  socket_id : id } ) );
                }

                socket.onmessage = function (ev) {
                    var json = JSON.parse( ev.data ) || {};
                    var signal = json.signal;
                    var body = json.body;

                    if( signal == 'entity' ){ scope.mapEntity( body ); }

                    if( signal == 'message' ){
                        M.toast({ html : body });
                    }

                    if( signal == 'tick' ){
                        var divs = document.querySelectorAll('.timestamp');
                        for(var i in divs){
                            if( divs[i].innerHTML ){
                            divs[i].innerHTML = new Date( body.time );
                            }
                        };
                    }
                };

                socket.onerror = function (err) { M.toast({ html : err }); };

            })
            .catch( err => {
                M.toast({ html : 'Simulation is not active.' });
                setTimeout( connectSocket , waited++ * 1000 );
            });

        };
        connectSocket();
    }
}
```

```
alight.directives.al.jsontotable = function(scope, elm, exp, env) {

    var data_path = env.takeAttr( 'data-path' );
    var jdata = scope[ data_path ];
    if( jdata.length <= 0 ){ return; }

    var tbody = document.createElement("tbody");
    var table = document.createElement("table");
    table.className = "table highlight";

    var colnames = Object.keys( jdata[0] );
    var tr = document.createElement("tr");
    colnames.map( c => {
        var th = document.createElement("th");
        th.innerHTML = c;
        tr.append( th );
    });
    tbody.append( tr );

    jdata.map( d => {
        var tr = document.createElement("tr");
        colnames.map( c => {
            var td = document.createElement("td");

            td.innerHTML = JSON.stringify( d[c] );

            tr.append( td );
        });
        tbody.append( tr );
    });

    table.append( tbody );
    elm.append( table );
};
```

```
<div al-entities page-size="20" page="0" entity-name-plural="datasources">
    <p> You requested {{page_size}} datasources but there
        are total {{count}}.</p>
    <ul>
        <li al-repeat="d in datasources">{{ d.id }} - {{d.name}}</li>
    </ul>
</div>
```

Code B.20: HTML and Javascript code using the `al-entities` created and `al-repeat` built-in directive. When Alight finds the `al-entities` directive, it will execute the Javascript code setting the scope accessible variables, and fetching entities. The response is stored in the scope too and `env.scan()` is invoked to spread the changes. Now that there are datasources in the scope, the `al-repeat` directive will iterate them rendering the list item for every datasource.

APPENDIX

C

Publications

C.1 Awards

"UD-Grupo Santander de InvestigaciÃşn award for SmartGrid 3.0: towards the SmartCity" by Universidad de Deusto âĂŞ Banco Santander (2015).

"Best Demo Award" in AAMAS 2014 Demo track by IFAAMAS (2014).

"Cloud Bilbao Award" by Ayuntamiento de Bilbao Cimubisa - Deusto (2014).

C.2 Indexed journal articles

Ander Pijoan Lamas, Oihane Kamara Esteban, Ainhoa Alonso Vicario, Cruz Enrique Borges Hernández. (2018) "Transport Choice Modeling for the Evaluation of New Transport Policies" In Sustainability. vol. 10. DOI: 10.3390/su10041230.

Oihane Kamara Esteban, Gorka Azkune, Ander Pijoan Lamas, Cruz Enrique Borges Hernández, Ainhoa Alonso Vicario, Diego López de Ipiña González de Artaza.

(2017) "MASSHA: An agent-based approach for human activity simulation in intelligent environments" In Pervasive and Mobile Computing. DOI: https://doi.org/10.1016/j.pmcj.2017.07

Oihane Kamara Esteban, Ander Pijoan Lamas, Ainhoa Alonso Vicario, Cruz Enrique Borges Hernández. (2017) "On-demand energy monitoring and response architecture in a ubiquitous world" In Personal and Ubiquitous Computing. vol. 21. p. 1-15. DOI: 10.1007/s00779-017-1014-4.

Diego Casado Mansilla, Ioannis Moschos, Oihane Kamara Esteban, Apostolos C. Tsolakis, Cruz Enrique Borges Hernández, Stelios Krinidis, Ane Irizar Arrieta, Konstantinos Kitsikoudis, Ander Pijoan Lamas, Dimitrios Tzovaras, Diego López de Ipiña Gonzóalez de Artaza. (2018) "A Human-centric & Context-aware IoT Framework for Enhancing Energy Efficiency in Buildings of Public Use" In IEEE Access. DOI: 10.1109/ACCESS.2018.2837141.

C.3 **Conference papers**

Oihane Kamara Esteban, Gorka Sorrosal Yarritu, Ander Pijoan Lamas, Tony Castillo Calzadilla, Xabier Iriarte López, Ana María Macarulla Arenaza, Cristina Martín Andonegui, Ainhoa Alonso Vicario, Cruz Enrique Borges Hernández. (2016) "Bridging the Gap between Real and Simulated Environments: an Agent-Based Hybrid Smart Home Simulator for Complex System" In Proceedings of the 13th IEEE International Conference on Ubiquitous Intelligence and Computing .

Ander Pijoan Lamas, Oihane Kamara Esteban, Cruz Enrique Borges Hernández, Iraia Oribe García, Cristina Martín Andonegui, Ainhoa Alonso Vicario. (2015) "Geosimulaciones: cuando un SIG no es suficiente" In Actas de las 9as Jornadas de SIG Libre.

Ander Pijoan Lamas, Oihane Kamara Esteban, Cruz Enrique Borges Hernández, Yoseba K. Penya Landaburu. (2014) "GIS and MAS tight coupling for Spatial Load Forecasting" In Proceedings of the 13th International Conference on Autonomous Agents and Multiagent Systems. p. 12.

Cruz Enrique Borges Hernández, Oihane Kamara Esteban, Ander Pijoan Lamas, Yoseba K. Penya Landaburu. (2014) "Multi-Agent GIS System for Improved Spatial

Load Forecasting (Demonstration)" In Proceedings of the 13th International Conference on Autonomous Agents and Multiagent Systems. p. 2.

Ander Pijoan Lamas, Cruz Enrique Borges Hernández, Yoseba K. Penya Landaburu, Ainhoa Alonso Vicario. (2014) "Uso de algoritmos de enrutamiento para el cálculo de indicadores de sostenibilidad" In Actas de las 8as Jornadas de SIG Libre. p. 24.

Cruz Enrique Borges Hernández, Ander Pijoan Lamas, Gorka Sorrosal Yarritu, Iraia Oribe García, Mikel González Astorga, Oihane Kamara Esteban. (2013) "Uso de fuentes de información geográfica voluntarias en proyectos de ingeniería" In Actas de las 7as Jornadas SIG Libre. p. 22.

Cruz Enrique Borges Hernández, Yoseba K. Penya Landaburu, Ander Pijoan Lamas. (2012) "Agent Based Spatial Load Forecasting" In Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems. p. 2.

C.4 **Book sections**

Ander Pijoan Lamas, Oihane Kamara Esteban, Iraia Oribe García, Ainhoa Alonso Vicario, Cruz Enrique Borges Hernández. (2017) "GTPlat: Geosimulation for Assessing the Application of Incentives to Transport Planning" In Advances in Intelligent Systems and Computing. DOI: 10.1007/978-3-319-62316-0_6.

Ander Pijoan Lamas, Iraia Oribe García, Oihane Kamara Esteban, Konstantinos Genikomsakis, Cruz Enrique Borges Hernández, Ainhoa Alonso Vicario. (2016) "Regression based emission models for vehicle contribution to climate change" In

Intelligent Transport Systems and Travel Behaviour. p. 47-63. DOI: 10.1007/978-3-319-43991-4_5.

Ander Pijoan Lamas, Oihane Kamara Esteban, Cruz Enrique Borges Hernández. (2015) "Environment Modelling for Spatial Load Forecasting" In Environments for Multi-Agent Systems IV. vol. 1. p. 188-206. DOI: 10.1007/978-3-319-23850-0_12.

Alessandro Ricci, Juan Antonio Rodriguez, Ander Pijoan Lamas, Franco Zambonelli. (2015) "Mixed Environments for MAS: Bringing Humans in the Loop" In Environments for Multi-Agent Systems IV. vol. 1. DOI: 10.1007/978-3-319-23850-0_4.

Ander Pijoan Lamas, Cruz Enrique Borges Hernández, Iraia Oribe García, Cristina Martín Andonegui, Ainhoa Alonso Vicario. (2015) "Agent Based Simulation for the Estimation of Sustainability Indicators" In Procedia Computer Science. vol. 51. p. 2943-2947. DOI: 10.1016/j.procs.2015.05.488.

Declaration

I, Ander Pijoan, herewith declare that this dissertation is my own original work, carried out as a doctoral student at the University of Deusto. All assistance received and notions from other sources have been identified as such, acknowledging their correspondent contributions and citing them properly.

This work contains no material which has been presented in identical or similar form to any examination board, except where due acknowledgement is made in the dissertation.

This dissertation was finished writing on July 22th, 2018.