FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Co-Simulation Architecture for Environmental Disturbances

Rafael Ricardo Damasceno

Mestrado Integrado em Engenharia Informática e Computação

Supervisor: Daniel Castro Silva

July 23, 2020

Co-Simulation Architecture for Environmental Disturbances

Rafael Ricardo Damasceno

Mestrado Integrado em Engenharia Informática e Computação

Approved in oral examination by the committee:

Chair: Prof. Rui Carlos Camacho de Sousa Ferreira da Silva
External Examiner: Prof. Paulo Jorge Pinto Leitão
Supervisor: Prof. Daniel Augusto Gama de Castro Silva

July 23, 2020

Abstract

As of late, the number and the diversity of autonomous vehicles used in various tasks have been increasing. A Platform was created in the Artificial Intelligence and Computer Science Laboratory to allow for the simulation of varying kinds of cooperative missions with a group of heterogeneous vehicles. Examples of such missions are detecting a fire, locating a pollution source or finding and following a vehicle. The Platform contains a Disturbances Manager, which allows for the simulation of environmental disturbances that may require vehicular intervention. Different kinds of disturbances and their characteristics must be taken into account, including spatial aspects like their size, location, multiplicity, and temporal ones, namely mobility and growth.

Currently, the aforementioned disturbances are simulated using Flight Simulator X (FSX), which serves as the core engine for The Platform. The Disturbances Manager was created to simulate disturbances that could not be simulated in FSX. However, it would be better to have dedicated simulators for certain types of environmental disturbances, as the Disturbances Manager cannot simulate some of them with the required higher realism and simulation fidelity.

The most adequate co-simulation architecture to successfully integrate external simulators for environmental disturbances, via the Disturbances Manager, was determined. An architecture to possibilitate this while also standardising the communication between components of The Platform was put into place using RabbitMQ.

The best suited external simulator for The Platform was searched. Fire simulators were given priority for FSX compatibility with their input data yet earthquake, tsunami and atmospheric dispersion simulators were also researched. ForeFire was chosen to simulate surface wildfire spread.

The co-simulation architecture was implemented using the previous choices. Data needed for the fire simulation was extracted from the one used by FSX. Some preprocessing was required, like converting land classification systems and interpolating wind values. A Python middleware facilitated the inclusion of the external simulator in The Platform's architecture and the communication between this simulator and the Disturbances Manager. A buffer mechanism for time synchronisation was implemented in the Disturbances Manager.

A test suite was created to evaluate and validate the implemented solution. The implemented solution was tested and validated in different wind and location scenarios using various metrics, like simulation speed and time drift.

The solution was found to be good enough to support 4x real-time simulation speeds in the initial stages of slow fire spread, 2x speeds thereafter and 1x for fast fire spread. Improvements can be made in the solution by addressing data compatibility issues that cause the simulation speed to deteriorate as the simulation time increases.

Keywords: co-simulation architectures, environmental disturbances

Resumo

Ultimamente, o número e a diversidade de veículos autónomos utilizados em várias tarefas têm vindo a aumentar. Foi criada uma Plataforma no Laboratório de Inteligência Artificial e Ciência de Computadores para permitir a simulação de vários tipos de missões cooperativas com um grupo de veículos heterogéneos. Exemplos de tais missões são detetar um incêndio, localizar uma fonte de poluição ou encontrar e seguir um veículo. A Plataforma contém um Gestor de Distúrbios, o que permite a simulação de distúrbios ambientais que podem requerer intervenção veicular. Diferentes tipos de distúrbios e as suas características devem ser tidos em conta, incluindo aspectos espaciais como o seu tamanho, localização, multiplicidade, e temporais, nomeadamente a mobilidade e o crescimento.

Actualmente, os distúrbios mencionados acima são simulados utilizando o Flight Simulator X (FSX), que serve como motor principal para A Plataforma. O Gestor de Distúrbios foi criado para simular distúrbios que não podiam ser simulados no FSX. No entanto, seria melhor ter simuladores dedicados para certos tipos de distúrbios ambientais, uma vez que o Gestor de Distúrbios não consegue simular alguns deles com o realismo e a fidelidade de simulação exigidos.

Foi determinada a arquitetura de co-simulação mais adequada para integrar com sucesso simuladores externos para distúrbios ambientais, através do Gestor de Distúrbios. Foi criada uma arquitetura que possibilitasse isto ao mesmo tempo que uniformizava a comunicação entre os componentes da Plataforma, utilizando o RabbitMQ.

Foi pesquisado o simulador externo mais adequado para A Plataforma. Foi dada prioridade aos simuladores de incêndio tendo em vista a compatibilidade do FSX com os seus dados de entrada, mas também foram pesquisados simuladores de sismos, tsunamis e dispersão atmosférica. O ForeFire foi escolhido para simular a propagação de incêndios florestais à superfície.

A arquitetura de co-simulação foi implementada utilizando as escolhas anteriores. Os dados necessários para a simulação do incêndio foram extraídos dos dados utilizados pelo FSX. Foi necessário algum pré-processamento, como a conversão de sistemas de classificação de terrenos e a interpolação de valores de vento. Um middleware em Python possibilitou a inclusão do simulador externo na arquitetura da Plataforma e a comunicação entre este simulador e o Gestor de Distúrbios, onde foi implementado um mecanismo de buffer para a sincronização temporal.

Foi criado um conjunto de testes para avaliar e validar a solução implementada. A solução implementada foi testada e validada em diferentes cenários de vento e localização, utilizando várias métricas, como velocidade de simulação e desvio temporal.

A solução foi considerada boa o suficiente para suportar inicialmente velocidades de simulação de 4x em propagações lentas e velocidades seguintes de 2x, e de 1x para propagações rápidas. Podem ser feitas melhorias na solução, abordando questões de compatibilidade de dados que causam a deterioração da velocidade de simulação à medida que o tempo de simulação aumenta.

Palavras-chave: arquiteturas de co-simulação, distúrbios ambientais

Acknowledgements

I would like to express my gratitude towards everyone that helped make this work possible.

Firstly, to Daniel Silva, my supervisor, for his dedication, organisation and helpfulness. His support and comprehensiveness were always remarkable and unparalleled, be it regarding work or any other matters.

Secondly, to Davide Costa, João Esteves and Tiago Carvalho, for enduring all of the endless meetings and issues we ran into together working on The Platform, as well as helping whenever our work overlapped.

Thirdly, to Jean-Baptiste Filippi, for his work with ForeFire and his support in understanding important aspects of this wildfire model.

Finally, to my friends, for providing an escape when the work seemed overwhelming, to my family, for always believing in me, and to my brother and my girlfriend.

Rafael Ricardo Damasceno

A ESTOUVACA

"deitada atravessada
na estrada
a malhada
vai ser atropelada
foi"

Alexandre O'Neill

Contents

List of Figures

List of Tables

Abbreviations

AHC	Analytic Hierarchy Process
AMQP	Advanced Message Queuing Protocol
API	Application Programming Interface
ATC	Air Traffic Control
CDL	Common Data Language
CLC	CORINE Land Cover
COTS	Commercial Off-the-shelf
CPU	Central Processing Unit
CSP	COTS Simulation Packages
CSV	Comma-separated Values
CT	Continuous Time
DDL	Disturbance Description Language
DDS	Data Distribution Service
DE	Discrete Event
DIS	Distributed Interactive Simulation
DoD	United States Department of Defense
DSEEP	Distributed Simulation Engineering and Execution Process
FEUP	Faculty of Engineering, University of Porto
FIPA	Foundation for Intelligent Physical Agents
FMI	Functional Mockup Language
FMU	Functional Mockup Unit
FOM	Federation Object Model
FSX	Microsoft's Flight Simulator X
GUI	Graphical User Interface
HLA	High Level Architecture
IEC	International Electrotechnical Commission
IEEE	Institute of Electrical and Electronics Engineers
IP	Internet Protocol
JMS	Java Message Service
ITEA	Information Technology for European Advancement
LIACC	Artificial Intelligence and Computer Science Laboratory
LVC	Live, Virtual and Constructive
MQTT	Message Queuing Telemetry Transport
netCDF	Network Common Data Form
ns-3	Network Simulator 3
NTP	Network Time Protocol
OMG	Object Management Group
OMT	Object Model Template

OpenDSS	Open Distribution System Simulator
PDU	Protocol Data Unit
PTC	Progressive Transform Coder
QMID	Quad Mesh Identifiers
QoS	Quality of Service
RPR	Real-time Platform Reference
RPR-FOM	Real-time Platform Reference Federation Object Model
RTI	Runtime Infrastructure
SDK	Software Development Kit
STOMP	Simple (or Streaming) Text Oriented Message Protocol
SOM	Simulation Object Model
SWOT	Strengths, Weaknesses, Opportunities and Threats
TCP	Transmission Control Protocol
TENA	Test and Training Enabling Architecture
UDP	User Datagram Protocol
UML	Unified Modeling Language
USA	United States of America
WSL	Windows Subsystem for Linux
XML	Extensible Markup Language

Chapter 1

Introduction

An overview of the present report is presented in this chapter. Firstly, the context and the motivation behind this work. Secondly, the detailing of its goals. Thirdly, explaining the methodologies and expected results stemming from its developments and lastly, an outline of the report's structure.

1.1 Context and Motivation

As of late, there is a growing number and diversity of autonomous vehicles [MarketsandMarkets, 2019]. Consequently, a Platform was developed in the Faculty of Engineering, University of Porto's (FEUP) Artificial Intelligence and Computer Science Laboratory (LIACC), using Microsoft's Flight Simulator X (FSX) as a starting point [Silva, 2011]. The main objective of the platform is to simulate autonomous vehicles cooperating in group missions like transportation, search and origin detection. However, as the frequency of environmental disturbances increases, there is a need for better suited methods to properly study their interaction with these vehicles, seeing that they might be able to detect certain disturbances and act accordingly [CRED, 2015].

Recent developments on this Platform include the addition of a disturbance simulation component via the Disturbances Manager of The Platform, which allowed the simulated vehicles to become aware of disturbances occurring near their location [Almeida, 2017]. These disturbances are simulated within FSX when it is able to simulate a given disturbance, and they are managed using the Disturbances Manager. However, because the FSX engine is originally intended to be used as a game, some simulation details aren't as interesting as one might expect, e.g. there's a visual effect for fires, but no propagation simulation. As a consequence of this, and because FSX does not either support the simulation of a given kind of disturbance or the realism and fidelity levels of the simulation are required to be higher for a certain kind of disturbance, the use of external simulators is required.

1.2 Goals

The main goal of this project is to answer the following research question: Is it possible to use an external simulator for simulating specific environmental disturbances inside The Platform, given a single co-simulation architecture or standard?

The present work aims to carefully choose an environmental disturbance simulator to be coupled through the Disturbances Manager, using a co-simulation architecture. The objective is to reliably and faithfully simulate environmental disturbances using a co-simulator properly suited to simulate those, and then have the disturbances in the simulated environment of The Platform. In order to achieve this, there are some requirements regarding time and space synchronisation, to guarantee that the disturbances are simulated within the same conditions that exist within The Platform.

1.3 Methodologies and Expected Results

The work begins with an exhaustive study of the literature on co-simulation and existing frameworks. Being able to pinpoint the advantages and disadvantages of any given co-simulation framework is essential to choose the most adequate one for the project. It is also important to understand if these frameworks are mere guidelines or are actually enforcing in nature, and to take into account the consequences they will have upon the following development.

Afterwards, the external simulators must be chosen. Certain traits, like the impact on the real-time performance of the platform, the available documentation, having an application programming interface (API), being free and open-source, available functionality, etc., will need to be taken into consideration and an evaluation must be done to find what are the best matches for this project for any given type of disturbance.

Finally, the development of a test suite is required in order to properly analyse and observe the performance impact that linking the external simulators had on the simulation occurring in The Platform. It should also be verified that the simulated disturbances are of a higher realism and fidelity than the ones that would be found in FSX, to understand that its inclusion was an addition to the quality of the overall simulation and not a deterrent. There's also the requirement of guaranteeing that the same data is being used in the external simulator and The Platform, which might require additional data synchronisation work. To achieve this, an analysis will be performed on aspects such as time drift, data acceptance/compatibility, simulation speed and information latency.

It is expected that the most adequate co-simulation framework and external simulators are chosen. Then, using the chosen framework, the chosen simulators are integrated into The Platform. The test suite should provide good insight into the performance and quality metrics of adding the external simulators to the platform, and the assessment should indicate that the added simulators do an overall better job at simulating the disturbances than The Platform itself ever could.

1.4 Document Structure

The next sections of this report are structured as follows:

Chapter 2 aims to provide some contextualisation on the concepts of The Platform, its simulation architecture, environmental disturbances and information on simulation architectures. Chapter 3 contains the state of the art for environmental disturbances simulation and co-simulation architectures. Chapter 4 describes the planned implementation steps, a brief risk analysis, the chosen architecture for implementing a solution and the technological choices. Chapter 5 depicts the solution used for implementing the projected architecture. Chapter 6 shows the performed tests on the implemented solution. Chapter 7 presents a summary of the report, states the conclusions reached and a summary of the possible future work.

Chapter 2

Contextualisation

This chapter will serve to contextualise the reader in regards to existing co-simulation standards and architectures, as well as the categorisation of the environmental disturbances that will be addressed in The Platform.

2.1 Co-Simulation Standards and Architectures

This section will outline the standards and architectures for distributed and co-simulation. These can either be simulation-oriented, or non-simulation-oriented. The ones presented first are simulation-oriented standards and architectures for co-simulation. This means that these have been thought of, from the ground up, with simulation and aspects relevant to them in mind. The last one, however, is not simulation-oriented.

Co-simulation can be defined as a composition of heterogeneous simulators used to simulate a global environment, with specialised simulators being used for different objectives [Gomes et al., 2018]. This can differ from distributed or parallel simulation, where the main purpose is simply to distribute a single simulation model over multiple machines, in order to achieve higher performance [Fujimoto, 2015].

2.1.1 Distributed Interactive Simulation

Distributed Interactive Simulation (DIS) is a protocol created in the mid-1980s by the United States Department of Defense (DoD) [Fullford, 1996]. It was primarily built with military operations in mind, but later became an Institute of Electrical and Electronics Engineers (IEEE) Standard, being utilised in other areas [IEEE C/SI, 2015].

The main component of DIS is the Protocol Data Unit (PDU), which is defined as the data messages which are traded between simulation applications [Hofer and Loper, 1995]. There are several domains with defined PDUs, which are called protocol families, including Entity Information/Interaction, Warfare, Logistics, Simulation Management, Distributed Emission Regeneration,

Radio Communications, Entity Management, Minefield, Synthetic Environment, Simulation Management with Reliability, Information Operations, Live Entity Information/Interaction, and Non-Real-Time protocol [IEEE C/SI, 2012]. There are, as of the latest revision of the standard, 72 of these network packets, which are then distributed through the simulation nodes. They are encapsulated in UDP/IP frames and distributed via broadcast, or over the Internet using IP multicasting [Fullford, 1996].

Even though there are relatively recent updates on the standards, DIS is limited in the scope of what it can bring as a simulation standard [IEEE C/SI, 2012, IEEE C/SI, 2015]. Because the PDU is the only well-defined building block for DIS, it did not achieve the interoperability that was required of higher-order war game models. At the time, DIS only supported real-time simulations (which was solved in newer versions of the standard) and the fact that the message structure was immutable might also become a problem. Because of this, a new generation DIS was developed, which is known as High Level Architecture [Fullford, 1996].

2.1.2 High Level Architecture

High Level Architecture (HLA) is a newer family of the DoD simulation common architecture [Dahmann et al., 1998a]. It also has since become a widely implemented IEEE standard [IEEE C/SI, 2010b].

HLA was built based on the premise that no individual simulation model can possibly satisfy all uses and users requirements. For this reason, its main purpose is to allow the reuse of the capabilities of different simulations, cutting down the cost and time required to bring together a new simulation environment for a new purpose [Dahmann et al., 1998a]. A functional view of HLA can be seen in Fig. 2.1.

There are three main terms used to describe elements in HLA: a federation, which is a set of models forming a bigger simulation; a federate, a member of a federation, which can represent a simple model or an aggregate simulation; and a federation execution, which is a simultaneous execution of federations [Dahmann et al., 1998b].

With these three terms in mind, it is possible to define the essential HLA triad:

- **HLA Rules**, principles which define the responsibilities of federates, ensuring proper interaction between them.

- **Object Model Template (OMT)**, a generic way to define the entities and their interactions in a federation. Federates have *Simulation Object Models* (SOM) and federations have *Federation Object Models* (FOM). They are a common representation in a federation or federation execution.

- **Interface Specification**, which defines the interface between the federates and the *Runtime Infrastructure* (RTI). The RTI is the base of the communication of simulation entities. However, it is not the RTI that is standardised, but the interface to it.

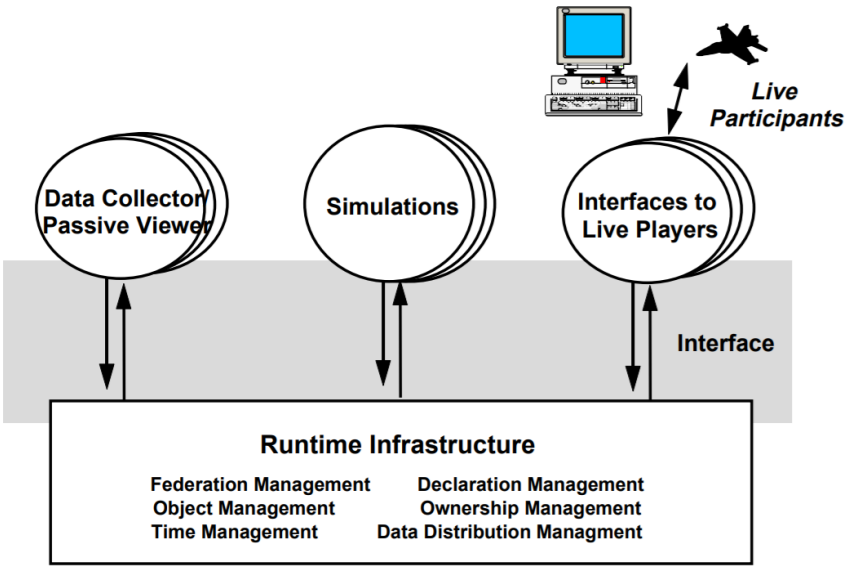


Figure 2.1: Functional View of an HLA Federation [Dahmann and Morse, 1998]

It is now possible to understand that in HLA, federates share a common RTI, which they communicate with through an interface. Federates belonging to the same simulation environment constitute a federation, and their SOM representation is common inside the federation. Multiple federations can be running atop the RTI. This differentiates HLA from DIS in the aspect that HLA allows defining new types of message contents, whereas in DIS has a strict set of types. If a certain simulation needs to communicate about things that are out of the scope of DIS, HLA is a good alternative.

The Federation Development and Execution Process was developed and updated to help with the user implementation of HLA [IEEE C/SI, 2003]. Dahmann and Morse reckon that the utility of HLA will be based on the availability of HLA tools that allow for easily maintaining such a simulation environment [Dahmann and Morse, 1998]. In particular, RTI software and object model tools are useful in the implementation of an HLA simulation. There are known implementations of the RTI, both commercial and non-commercial, such as MAK RTI[1], CERTI [2].

2.1.3 Test and Training Enabling Architecture

Test and Training Enabling Architecture[3] (TENA) is a newer architecture by the DoD for simulating activities that are live, virtual and constructive (LVC) simulations [Noseworthy, 2008].

The concepts of LVC are as follows: live means the ability to support real, physical assets in the simulation; virtual, being able to support simulated physical assets; and constructive, simulation environments in which models of physical assets are used.

[1]MARK RTI is available at https://www.mak.com/products/link/mak-rti.
[2]CERTI is available at https://savannah.nongnu.org/projects/certi/.
[3]TENA is available at https://www.tena-sda.org/.

The critique of HLA and DIS by TENA developers is that simulations built on top of the aforementioned are inherently virtual and constructive. Because they're not real, the flow of time is not constrained by reality and the cost of failure is generally low, unlike the simulations with real components in the war scenarios, which involve real machinery and operators.

To that end and to support the kinds of LVC simulations required of the DoD, TENA was developed, which enforces the form of data exchange to reduce faults in the simulation, and therefore its costs. It has at its core the TENA Middleware, a tool that uses Unified Modeling Language (UML) to automatically generate high-level, model-driven code. The object model needs to be created within the middleware and is regarded as the most important task of the development [Hudgins and Secondline, 2018]. The general TENA architecture can be observed in Fig. 2.2.

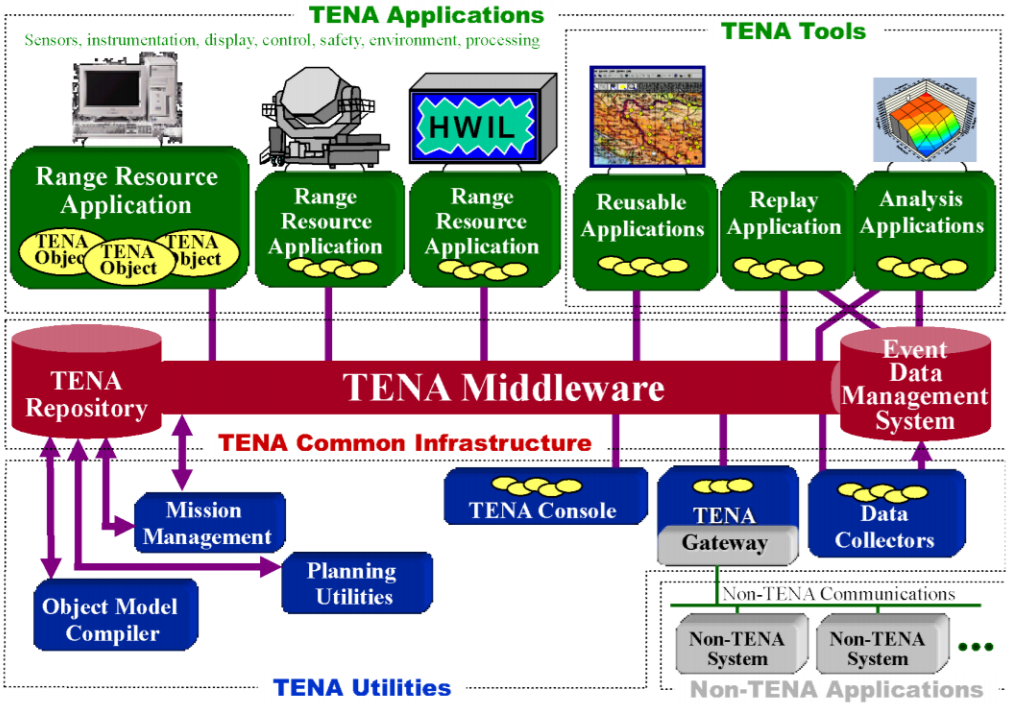


Figure 2.2: TENA Architecture [Hudgins and Secondline, 2018]

At this time, the TENA Middleware is in full control of the DoD and, to be able to use it, one is required to create an account on their platform and abide by United States Government sanctioned usage. The creation of the account and usage of the middleware is free, however.

2.1.4 Functional Mockup Interface

The Functional Mockup Interface[4] (FMI) is a tool developed by the coordination of European industry and academic researchers, under the Information Technology for European Advancement (ITEA2) project MODELISAR [Chombart, 2012]. It serves as a tool independent standard to exchange dynamic models and for co-simulation [Blochwitz et al., 2011].

[4]FMI is available at https://fmi-standard.org/.

The context of the development of FMI was that, at the time, the model exchange between simulation tools was largely done through low-level interfaces available in modelling languages, which require much effort to be supported in a tool.

FMI defines two main modes of operation: FMI for Model Exchange and FMI for Co-simulation. In FMI for Model Exchange, the main idea is to have a generation of C-Code which can later be used as input/output blocks in other simulation environments. In FMI for Co-simulation, the participant sub-systems communicate using discrete communication points. However, when not communicating, they are each running their own algorithms. Data exchange and time management are done by so-called master algorithms, which in turn manage the slaves.

FMI components are Functional Mockup Units (FMU), which consist of a *.fmu* zip-file. This file contains an Extensible Markup Language (XML) file that defines what FMU variables are exposed to the environment it is inserted in and another one to define the slave capabilities. It also contains C functions in either source or binary code, both for using model exchange and for co-simulation, which in the latter case the functions serve communication purposes, for instance, establishment, step advancement and data exchange.

2.1.5 Data Distribution Service

Data Distribution Service (DDS) is a protocol in which data is distributed using a publish-subscribe model developed by the Object Management Group (OMG) [OMG, 2015, Pardo-Castellote, 2003]. It is highly performant, oriented for real-time systems. Nodes that produce content (publishers) can send a message of a certain topic, and then nodes that receive content (subscribers) seamlessly receive the content to topics they have subscribed. Nodes can be publishers and subscribers at the same time. It also implements Quality of Service (QoS) peer discovery mechanisms.

It is used in multiple platforms and it has implementations in various programming languages, which makes it useful for models who are also in the same situation [Madden and Glaab, 2017].

The only downside would be that because it is not specifically simulation-oriented, an aspect that is essential in simulation, time management, is not specified as part of DDS. To counter this, one would have to implement their own solution or use other existing synchronisation tools that can accomplish the same objective.

2.2 The Platform

As mentioned in section 1.1, The Platform is a tool to simulate missions of heterogeneous autonomous vehicles. It was developed in FEUP's LIACC. The missions to be simulated would be cases where it would be too dangerous for humans to intervene in such a situation, or because the cost of this kind of missions is too high. The Platform's initial architecture, as proposed by Silva, can be seen in Fig. 2.3. FSX is the simulation engine of The Platform. Its main point of interaction with the user is the Control Panel, where one can configure the simulation environment and monitor its execution. There are also Air Traffic Control (ATC) Agents, which are responsible for managing traffic within a given area or for a type of vehicle. These vehicles are represented

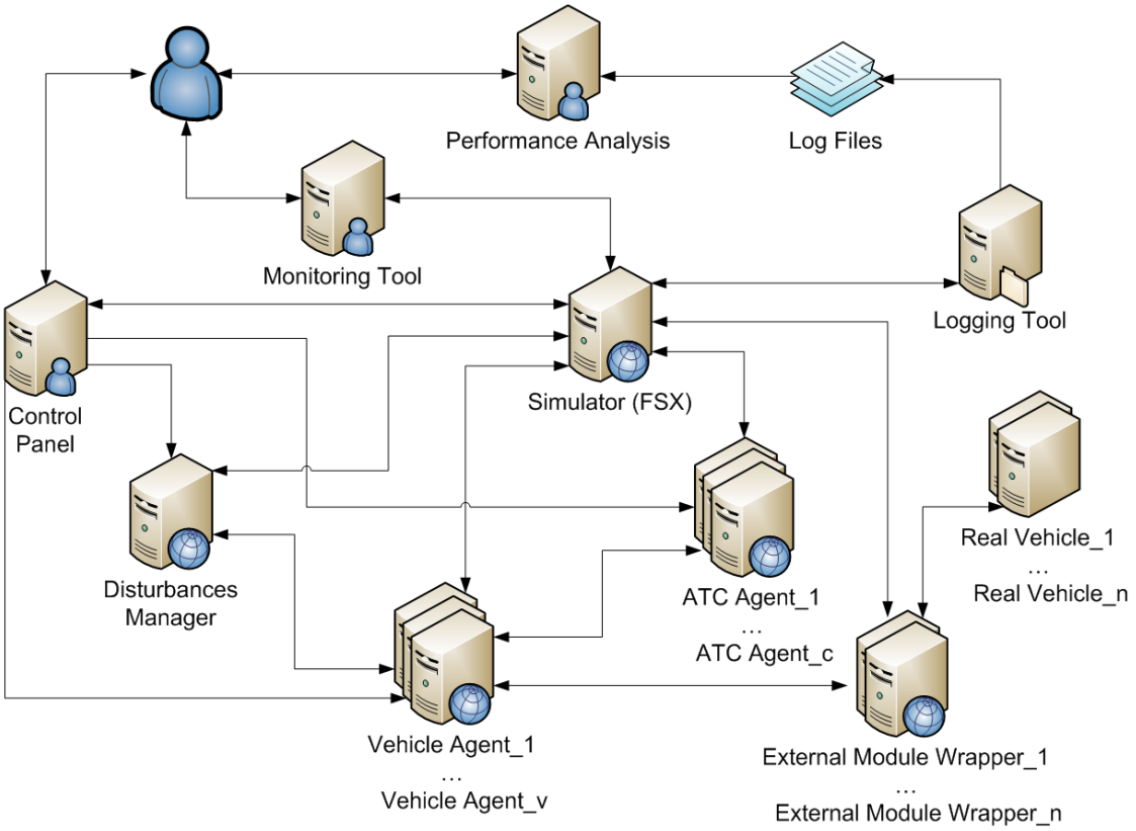


Figure 2.3: The Platform's Initial Architecture [Silva, 2011]

through the Vehicle Agents. Then, with the Log Files generated by the Logging Tool, we can use the Performance Analysis module to assess the current simulation performance. Finally, and perhaps the most important module for the project at hands, there's the Disturbances Manager, created to manage the disturbances not able to be within FSX. Most of the connection between The Platform's modules and the simulator is done through the SimConnect Software Development Kit (SDK), which can be used to create add-on components for Microsoft ESP-based simulators [Microsoft Corporation, 2008b, Microsoft Corporation, 2008a]. That is the case for Microsoft's own FSX. For the communication between agents, AgentService was used, which allows for Foundation for Intelligent Physical Agents (FIPA) compliant message exchange [Vecchiola et al., 2008]. The Control Panel graphical user interface (GUI) was visually revamped by the work of Esteves, but the functionality was retained [Esteves, 2020]. Figure 2.4 shows the GUI in its current state.

Together with The Platform, Silva proposed four XML-based dialects to define aspects related to the mission to be performed [Silva, 2011]. The four dialects are related to the static and dynamic properties of the mission definition. For the team, there's a dialect to define static items, the vehicles and their characteristics, and a dynamic one to define the missions they perform. In terms of the scenario, there's a dialect to define its static elements, such as bases of operation and no-fly areas and one to define dynamic features of the scenario. This last one is being further described

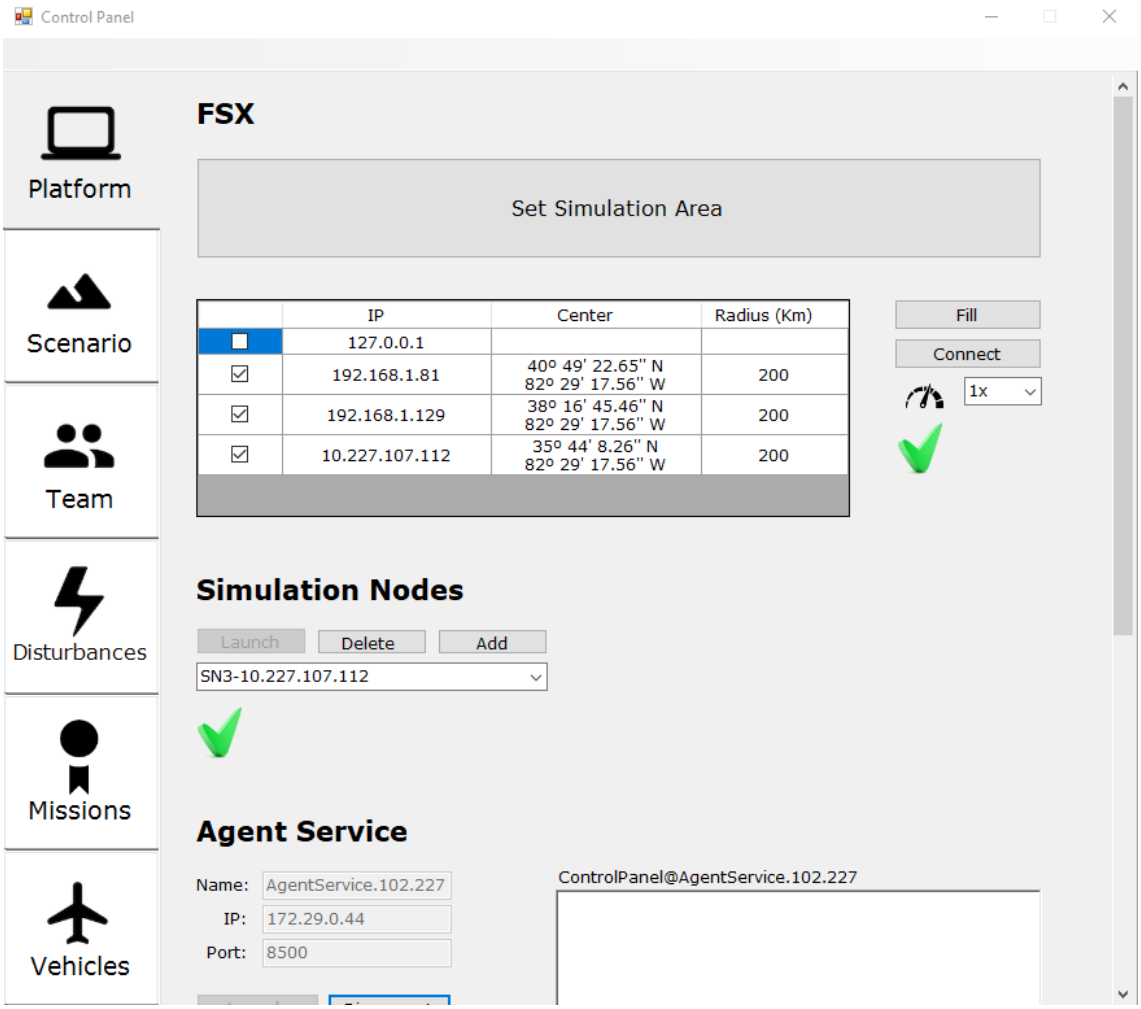


Figure 2.4: The Control Panel GUI [Esteves, 2020]

next because of its significance for this work.

To have a way to define the disturbances in an adjustable manner, the Disturbance Description Language (DDL) was created [Silva et al., 2016]. It allows for a flexible definition of the disturbances existing in The Platform. A diagram of DDL's schema, with the most recent additions the language from Almeida highlighted in blue boxes, can be seen in Fig. 2.5. One or more disturbances can be described at a time. DDL allows for the specification of the disturbance's nature, be it naturally occurring or man-made. It also allows for the definition of its location and time, which can be set to specific ones or also random values in a given range, like an area or time interval. The multiplicity and evolution of the disturbance are also detailed. It can be stationary, move, or grow. DDL allows for the specification of the evolution of the disturbance, using its *distSize* element, and the readings for sensors associated with a disturbance's component using the *generatedReading* element. However, should a co-simulator be successfully coupled, some of these fields would be set as custom so as to allow these parameters to be retrieved from the external simulator. Another important field is the component's *medium*, which refer to land, air, water and underwater boolean

values. This matches the reality of The Platform, in which the simulated vehicles can usually travel through one of these mediums, even though DDS was made for general usage.

2.3 Environmental Disturbances

Almeida outlined several types of environmental disturbances that were considered to be relevant to be represented in The Platform [Almeida, 2017]. The term *environmental disturbance* was chosen because *natural disaster* usually refers to non-man-made disasters, and the former was chosen to refer to every kind of disaster. This work served as a basis for the following tables that outline the varying kinds of environmental disturbances. However, several changes were made to the existing classification and groupings of the disturbances, as some of the existing ones seemed redundant or too specific. The work of Alexander was also used as a guideline for the types and categorisation of natural disasters [Alexander, 1993]. In his work, there are three main categories of natural disasters: earthquakes and volcanoes, atmospheric and hydrological hazards, and disasters and the land surface. These categories suggest of tectonics, fluids and surfaces, respectively, and are grouped as such because of the geophysical agent that causes those disasters, e.g. tsunamis are in the earthquakes and volcanoes category. However, because environmental disturbances include non-natural disasters, a geophysical agent aggregation makes less sense than it would if solely talking about natural disasters, seeing that man-made disasters would all be relegated to their own category. Another important aspect is that this listing is made in accordance with how The Platform works and how DDL is specified. The disturbances are defined there according to the medium in which they exist. Also, amphibious agents are not yet simulated. For these reasons and because it is, in a sense, suggested by the original natural disaster triad, the environmental disturbances are then categorised by the medium in which they occur. Tectonics and surfaces come together to represent land disturbances, while fluids get separated into atmospheric and waterbody disturbances.

The following tables include the name of the disturbance, its origin (*ManMade*, *Natural* or *Any*), the kinds of sensors that might be of interest in obtaining information regarding the disturbance, and if there is a growth rate associated with it. This is necessary for specifying its size evolution using DDL. They all also have a seriousness degree, i.e. the varying importance attributed to different occurrences of the same disturbance. Phenomena might not be directly stated in any of the tables; however, some other listed phenomena might represent it in a broader way or sense. The opposite situation can also happen, i.e. a disturbance that could probably be contained by another is listed; in this situation, it might be the case that the more specific disturbance is subject of more interest because of the effects it might have in the simulation, in the context of The Platform.

The land environmental disturbances (Table 2.1) groups events that occur because of and alter the terrain itself. Disturbances like vehicular accidents, which were previously on this table, were relocated since they're something that happen on land, but not within the land itself. Landslides

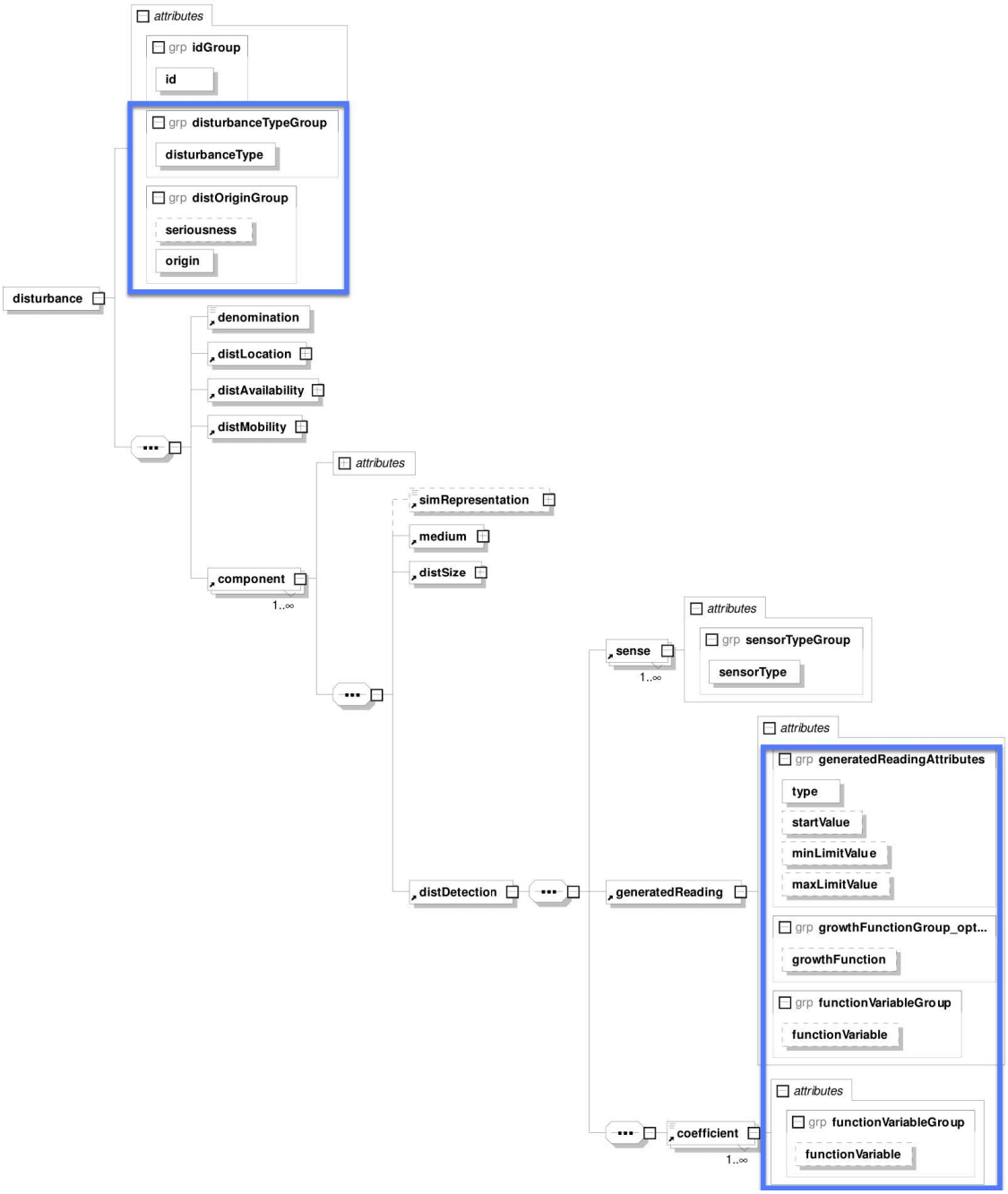


Figure 2.5: DDS Schema Diagram [Almeida, 2017]

refer to its many forms of manifestation, as is the case of avalanches. As for subsidence, it includes disturbances like sinkholes and cave-ins. The distinction between wildfire and urban fire is justified in the different way that their propagation is usually modelled and how it is combated.

Table 2.1: Land Environmental Disturbances

Name	Origin	Sensor Types	Growth Rate
Landslide	Natural	CameraIntensity; Microphone	yes
Subsidence	Natural	CameraIntensity; Microphone	yes
Earthquake	Natural	CameraIntensity; Microphone	yes
Flood	Natural	CameraIntensity	yes
Volcano Eruption	Natural	CameraIntensity; IRCameraIntensity; Temperature; Microphone	yes
Wildfire	Any	CameraIntensity; Chemical; IRCameraIntensity; Temperature	yes
Urban Fire	ManMade	CameraIntensity; Chemical; IRCameraIntensity; Temperature	yes

The waterbody environmental disturbances (Table 2.2) consist of disturbances occurring at a waterbody. Changes from the previous work include the aggregation of oil spills and chemical spills into a single disturbance since oil can be considered a chemical and they are both spills. Glacier hazards include all sorts of disturbances that can be caused by glaciers, as is the case of iceberg separations, for example. Strong waves include extreme cases, like tsunamis.

Table 2.2: Waterbody Environmental Disturbances

Name	Origin	Sensor Types	Growth Rate
Strong Waves	Natural	CameraIntensity; Microphone	yes
Chemical Spill	ManMade	CameraIntensity; Chemical	yes
Hydrothermal Vent	Natural	CameraIntensity; IRCameraIntensity; Microphone; Temperature	yes
Submarine Volcano Eruption	Natural	CameraIntensity; Chemical; IRCamera-Intensity; Microphone; Temperature	yes
Glacier Hazards	Natural	CameraIntensity; IRCameraIntensity; Microphone; Temperature	yes

The atmospheric environmental disturbances (Table 2.3) contains events that happen in the atmosphere. These are the disturbances that would be commonly referred to as happening in the air. They were renamed from *storm disturbances* in order to follow the new medium categorisation. For the tropical thunderstorms, they can be designated as hurricanes or typhoons, depending on their location. As for tornadoes, they also comprise similar phenomenon, e.g. dust devils, which are simply special manifestations of the former. Storm is a very broad disturbance which can

represent high persistence of winds or precipitation. It also represents some other, more specific disturbances that aren't otherwise mentioned, like sandstorms. However, other specific manifestations of storms, as is the case of blizzards and thunderstorms, exist in this listing because they pose great danger in aviation [Evans, 2014].

Table 2.3: Atmospheric Environmental Disturbances

Name	Origin	Sensor Types	Growth Rate
Storm	Natural	CameraIntensity; HumiditySensor; AirPressure; WindSpeed; Microphone	yes
Hail	Natural	CameraIntensity; IRCameraIntensity; HumiditySensor; AirPressure; WindSpeed; Microphone	yes
Blizzard	Natural	CameraIntensity; IRCameraIntensity; HumiditySensor; AirPressure; WindSpeed	yes
Fog	Natural	CameraIntensity; HumiditySensor; AirPressure	yes
Thunderstorm	Natural	CameraIntensity; HumiditySensor; AirPressure; WindSpeed; ElectricField; MagneticField; Microphone	yes
Tornado	Natural	CameraIntensity; AirPressure; WindSpeed; Microphone	yes
Tropical Cyclone	Natural	CameraIntensity; AirPressure; WindSpeed; Microphone	yes

The last category, assorted environmental disturbances (Table 2.4), encloses occasions that would not fit directly within a table above. Most of these disturbances happen independently of the conditions of the previous categories, even though they might happen within their mediums. They fit the new flock disturbance, a large group of birds, which is highly relevant and dangerous for aviation [Allan, 2000], causing a potential bird strike. The specific relevance it might have in the context of the simulated aerial vehicles is the reason it is differentiated from an animal infestation. The chemical contamination disturbance is included here as a broad manifestation of this event. It differs from the chemical spill, a specific case of chemical contamination, where the waterbody is severely affected by the spill. The contamination can refer to cases that would otherwise not be included, like an ammonia leak in a lab, or high CO_2 concentration in a room.

This categorisation and listing of disturbances should in no way be taken as exhaustive. Even though it is obviously as complete as it can be, it is always to be seen through the optics of The Platform's context, where certain events are of higher interest for simulation scenarios.

Accordingly, the sensor types indicated are the same from Almeida [Almeida, 2017]. These are sensors that can be included in the simulated unmanned vehicles. They are the following:

- AirPressure - An air pressure sensor, which outputs air pressure data, for example, in standard atmosphere (atm);

Table 2.4: Assorted Environmental Disturbances

Name	Origin	Sensor Types	Growth Rate
Animal Infestation	Natural	CameraIntensity	yes
Extreme Temperature	Natural	Temperature	yes
Chemical Contamination	ManMade	Chemical	yes
Radiation	ManMade	GeigerCounter	yes
Vehicular Accident	ManMade	CameraBinary	no
Lost/Fleeing Being	ManMade	CameraBinary; Microphone	no
Flock	Natural	CameraIntensity; IRCameraIntensity	no

- CameraBinary - A camera, which reads true or false, according to whether the disturbance in question can be observed. It will be calculated by whether or not the disturbance is within the agent's field of view;
- CameraIntensity - A camera, which reads an observed disaster normalised estimate intensity. A heuristic will be applied to calculate the intensity reading. It simulates results that a smart-camera, applying computer vision and/or machine learning algorithms to determine intensity values from a disturbance would yield;
- IRCameraIntensity - A camera, which reads an observed disaster normalised estimate intensity via infrared imaging. A heuristic will be applied to calculate the intensity reading;
- Chemical - A generic chemical sensor, which may convey readings in a specified concentration unit. A common use for this sensor is a CO_2 sensor;
- ElectricField - Measures the intensity of affecting electric fields, typically in newtons per coulomb (N/C);
- MagneticField - Gauges the intensity of affecting magnetic fields, typically in teslas (T);
- GeigerCounter - Indicates radiation intensity, usually measured in sievert (Sv);
- HumiditySensor - Measures the relative humidity of the air, expressed in a percentage;
- Microphone - Listens to sound waves. It is not limited to regular microphones, but can also mean hydrophones or other sensors that detect sound, independently of the medium. The loudness is commonly expressed in decibels (dB);
- Temperature - A sensor which reads the temperature, frequently in Celsius (^{o}C);
- WindSpeed - Measures the wind speed, generally in meters per second (m/s).

Chapter 3

State of the Art

This section describes the researched state of the art relating to co-simulation architectures, implementations of co-simulation platforms and natural disturbances simulators.

3.1 Co-Simulation Standards and Architectures

A good starting point for the research in what are the current challenges in co-simulation are the works of Gomes et al. [Gomes et al., 2017, Gomes et al., 2018]. In summary, it is stated that the increase in complex systems leads to a need for coupling different simulation models. Therefore, an exhaustive search was performed on state of the art of the previous five years, in search of generic approaches for co-simulation. There are two main standards outlined: the Functional Mock-up Interface (FMI) standard, and the High Level Architecture (HLA) standard. The former is said to be used in Continuous Time (CT) simulations, while the latter is used in Discrete Event (DE) ones. The one more carefully detailed is FMI, which was found to be used more than HLA recently in the researched environment. One advantage pointed out for FMI was that it allowed communication without revealing the IP behind a certain model, which was relevant for the mentioned industries. However, it was found that the aforementioned standards are limited in hybrid co-simulation, meaning that both CT and DE simulations are trying to be coupled. There are some proposed techniques and standard extensions to attenuate these shortcomings [Awais et al., 2013, Tavella et al., 2016].

Bauer and Van Duijsen outline two possible solutions for integrated simulations: parameter/-data exchange or co-simulation [Bauer and Van Duijsen, 2005]. Parameter/Data exchange is not relevant for this work as the simulated models are not equivalent in any way. Thus, the pointed out relevant solutions for co-simulation are Application Programming Interfaces (API) or Inter-process interfaces. However, no standard is mentioned in this work.

Fujimoto points out that distributed simulation faces many challenges, such as, but not restricted to, time management or distributing information between simulators efficiently [Fujimoto,

2015, Fujimoto, 2016]. Since interoperability has since been another of the goals in this area, standards to interconnect simulators have been created, such as HLA and DIS.

Lasnier et al. have some information about the HLA implementation and terminology [Lasnier et al., 2013]. The entire simulated system is a *federation*, which is composed of *federates*, these being the various simulation entities belonging to the system. These federates will need to communicate through a Run-Time Infrastructure (RTI). It points out that HLA defines three main characteristics: an interface specification for a set of services that enable federate management, an object model template for standardisation of federate communication, and rules that describe federate and federation responsibilities. Another notable thing in this work is the use of CERTI[1], an open-source HLA-compliant RTI.

Madden and Glaab delve into a different approach for running distributed simulations [Madden and Glaab, 2017]. Having previously used HLA and DIS, the researchers tried to implement a new simulation architecture based on the Distributed Data Service (DDS) specification, developed by the Object Managing Group, in a cloud setting [OMG, 2015]. It presented good results in providing data exchange in different network topologies (across same subnet, different subnets and from their intranet to the cloud provider). It was also found that implementation in various programming languages existed for this. As DDS is a specification for data exchange between real-time applications and has configurable quality of service (QoS), no-cost implementations, and includes interoperability wire protocols to deal with different vendor implementations, it was expected that it would be good for a simulation environment. However, because DDS is not a simulation framework, it lacks an important aspect which is time management. People who wish to use DDS will have to rely on other services, like Network Time Protocol (NTP), or implement their own solution. Another issue that DDS presents is that extensions to the DDS protocol are usually proprietary and thus, not interoperable.

Schweiger et al. use the Delphi method and performed a quantitative analysis of the strengths, weaknesses, opportunities and threats (SWOT) of co-simulation using the Analytic Hierarchy Process (AHP) [Schweiger et al., 2018, Okoli and Pawlowski, 2004]. However, it is mentioned that FMI is the prime candidate to become the industry and academy standard, going so far as to being included as a specific question in the Delphi study.

With this in mind, a matrix to outline the traits of the different standards and architectures is presented in Table 3.1. The analysed traits are ones that are relevant for this work, such as those important for simulation, performance, communication security and the overall usage of the standard or architecture. One big takeaway is that besides TENA, only an ad hoc solution might be able to achieve over the internet simulation right away, because the other standards, by themselves, are made to run on a single machine. Also, only an ad hoc solution might possibilitate a secure communication channel.

[1] CERTI is available at https://savannah.nongnu.org/projects/certi.

Table 3.1: Trait Matrix for Standards and Architectures. The symbol '?' denotes the inability in understanding if the trait is present or not on a given standard or architecture.

Traits	DIS	HLA	TENA	FMI	DDS	Ad hoc
Simulation-oriented	Yes	Yes	Yes	Yes	No	?
Run-time infrastructure	No	Yes	Yes	Yes	No	?
Synchronisation	No	Yes	Yes	Yes	No	?
Hot-plugging	No	No	?	?	Yes	?
High throughput	No	?	?	?	Yes	?
Over the internet	No	No	Yes	No	No	?
Secure communication	No	No	?	No	No	?
Widespread usage	Yes	Yes	No	Yes	Yes	Yes

3.2 Related Work

Králíček uses the aforementioned CERTI RTI and builds a plugin for FSX, to achieve HLA co-simulation [Králíček, 2011]. This is very relevant to the present work. The plugin is compiled using FSX's SDK, SimConnect, an FSX interface, and CERTI. It also takes into consideration different federation object models, like Real-time Platform Reference FOM (RPR-FOM). Advantages and disadvantages between the two are detailed. However, as we won't be simulating aircraft and their flight, these pros and cons might differ from what they would be in this application scenario.

Bian et al. developed a co-simulation platform using OPAL-RT[2] and OPNET[3], communication and power system simulators, respectively, for analysing smart grid performance [Bian et al., 2015]. The integration between these two tools was performed under MATLAB®. It is concluded that the co-simulation platform was successful in analysing communication failures on smart grid operation.

Schloegl et al. tried to achieve a classification schema for co-simulation in energy [Schloegl et al., 2015]. It also makes mention of mosaik[4], a Smart Grid co-simulation framework that allows one to reuse and combine existing simulation models and simulators to create large-scale Smart Grid scenarios. It is noted that an automated process to treat and analyse co-simulation platforms that consist of mainly black-box models must be developed, in order to achieve integrated and interdisciplinary energy systems.

Melman et al. present a distributed simulation framework for the investigation of autonomous underwater vehicles' real-time behaviour using their own implementation [Melman et al., 2015]. It consists of three layers: a server, which contains a supervisor and a database; managers, which contain the different simulation modules, and allow the data exchange with other modules and the server; and an interface, the control panel through which a user can control the simulation execution. The results regarded the architectural concepts as positive. Future improvements include the framework integration in an event-oriented software environment to control the vehicles.

[2]OPAL-RT is available at https://www.opal-rt.com/.
[3]OPNET is available at http://opnetprojects.com/opnet-network-simulator/.
[4]Mosaik is available at https://mosaik.offis.de/.

Cicirelli et al. implement a multi-agent system spacial environment architecture using THEATRE, an HLA-based agent infrastructure [Cicirelli et al., 2009, Cicirelli et al., 2015]. The system works using actor agents and messages, like common agent behaviour. The borders of the simulated space work like such: there's always a local replica of the border area of the remote environment, and vice-versa; when an agent is in the border of the remote environment, it gets in the locally replicated border area; when it crosses, it becomes part of the local world and is being replicated locally in the remote area of the remote environment. The results for the shared spatial environment are regarded as good, by allowing more parallelism and simpler territory management. Respective to the architecture, it is considered that benefits were provided when compared to not using any platform.

Brito et al. use HLA to implement a distributed simulation platform using several commercial-of-the-self (COTS) simulation packages (CSP), like Ptolemy II[5], SystemC[6], OMNeT++[7], Veins[8] and Stage[9], and physical robots [Brito et al., 2015]. The general architecture of the platform can be seen in Fig. 3.1. Ptolemy II is used to model the different Embedded Systems; OMNeT++ simulates network communication; SystemC is used to simulate circuitry; Veins as a way to simulate vehicle-to-everything communication; Stage, for simulated robots, and actual robots are used on the side to interact with the system. It is expressed that the platform was successfully applied to the five different experimental scenarios that are outlined in this work.

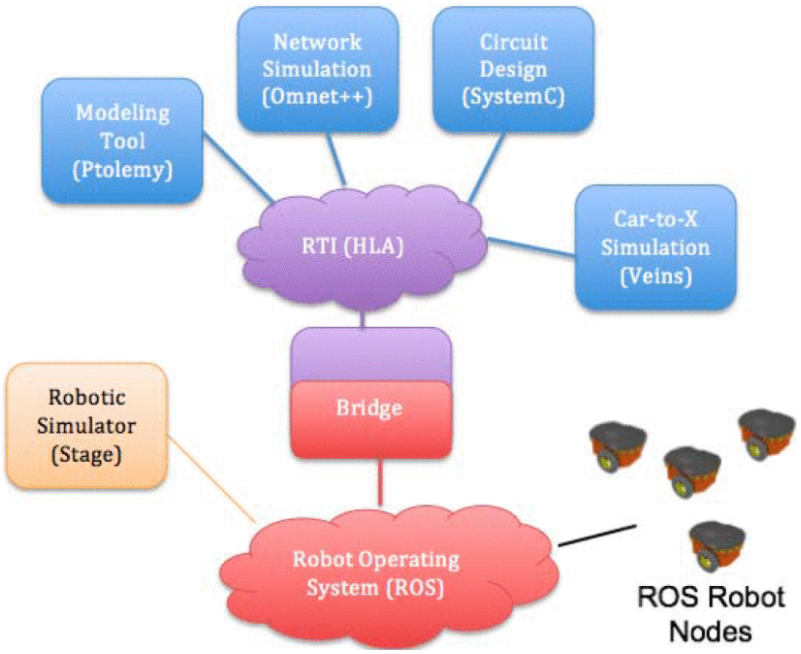


Figure 3.1: General Architecture in Brito et al. [Brito et al., 2015]

[5]Ptolemy II is available at https://ptolemy.berkeley.edu/ptolemyII/index.htm.
[6]SystemC is available at https://www.accellera.org/downloads/standards/systemc.
[7]OMNeT++ is available at https://omnetpp.org/.
[8]Veins is available at https://veins.car2x.org/.
[9]Stage is available at http://wiki.ros.org/stage.

Albagli et al. developed a smart grid framework co-simulation using HLA architecture [Albagli et al., 2016]. It is mentioned that a good approach to co-simulation is to develop a layer that deals with synchronisation and data transfer and then use a given framework to control the flow of information. An important referenced aspect is the standard IEEE 1516.3-2003 , which describes the best practices and procedures to create distributed simulation and execution (DSEEP) [IEEE C/SI, 2003]. This standard was, however, superseded by IEEE 1730-2010, which provided insight onto other 3 different architectures, like Distributed Interactive Simulation (DIS), HLA, and Test and Training Enabling Architecture (TENA) [IEEE C/SI, 2010a]. Because this work used homogeneous simulation agents, a common ontology was proposed to assist the HLA implementation, something that is not part of the standard requirements. However, it is not relevant in the project at hand, as simulators will have vastly different models and purposes. The usage of the frameworks in the implementation of the simulation environment and the common ontology for the simulators was successful. The results of the studied network configurations were not as positive since the simulated communication network leads to high simulation time.

Hong et al. implemented the occupant behaviour required of energetic simulations to be used with FMI, creating an FMU which contains the respective XML schema, interface, data model and solvers [Hong et al., 2016]. EnergyPlus is a simulation tool used frequently in energy simulation and it has support for the FMI standard. For that reason, it was used to test the developed FMU. FMI was also used by Dols et al., which coupled multizone airflow and contaminant transport software CONTAM[10] with EnergyPlus[11] [Dols et al., 2016]. The authors concluded that the developed model was flexible and interoperable and would aid in the detection of discrepancies between simulated and actual building energy use.

Manbachi et al. have a different approach to the co-simulation architecture [Manbachi et al., 2016]. It studies smart grid volt-VAR optimisation using the International Electrotechnical Commission (IEC) 61850 Manufacturing Message Specification, a standard which defines communication protocols for intelligent devices at electronic stations using TCP/IP [TC 57, 2020]. It distributes messages to the control components of the simulators using Distributed Network Protocol 3 [Majdalawieh et al., 2006]. The paper concludes that the IEC 61850 protocol can be of much help in volt-VAR optimisation and that the implemented real-time co-simulation platform could perform all the required tasks to achieve it.

D'Angelo et al. give an idea on how to simulate the Internet of Things, distributed and interconnected in its nature [D'Angelo et al., 2016]. It is claimed that an approach to this problem is to simulate CPU cores, CPUs or hosts that are parallelised and interconnected. However, due to the massive extension of a real Internet of Things network, it isn't feasible to simulate that many execution units. Hence, a multi-level simulation approach is proposed, using GAIA/ARTIS[12] for

[10] CONTAM is available at https://www.nist.gov/services-resources/software/contam.
[11] EnergyPlus is available at https://energyplus.net/.
[12] GAIA/ARTIS is available at https://pads.cs.unibo.it/doku.php?id=pads:download.

high-level simulation and, when more detail is needed, then tools like OMNeT++, Network Simulator 3 (ns-3)[13] and SUMO[14] are used.

Morakinyo et al. studied the effect of different roof types on the temperature and cooling demand [Morakinyo et al., 2017]. EnergyPlus and ENVI-met[15] were the two simulation tools used. However, they simply performed the data exchange between the models using the output of ENVI-met and creating a modified file that is accepted as the input in EnergyPlus, using no simulation standard. Results do not comment on the impact of the used architecture, with the main takeaway being that "the intent of green-roof installation should be a determining factor for the type and spatial extent to be implemented". It is also mentioned that these results shouldn't be taken into account in a vacuum, but rather with the results of the related work.

Garau et al. evaluate smart grid communication technologies [Garau et al., 2017]. There is mention of a tool, EPOCHS[16], which implements an RTI based on HLA. However, the solution used other tools, Open Distribution System Simulator (OpenDSS)[17] as the power system simulator, and ns-3 as the network simulator. MATLAB® was used as the RTI for this simulation, where distribution/energy management systems models were implemented. Thus, the simulation didn't follow any particular standard but is closely related to HLA. The results mention that the used co-simulation tools, such as the ones proposed by the authors, were necessary to be able to make any proper simulations regarding the matter. It was concluded that wireless technologies such as WiMAX, Wi-Fi and LTE meet the necessary smart-grid requirements for IEC 61850 [TC 57, 2020], even if additional work might be needed.

Cellura et al. integrate a building simulation and a Life Cycle Assessment (LCA) tool using simple text database files, in order to study the energy and environmental life cycle of buildings [Cellura et al., 2017]. The research was deemed by the authors as one of the first to study the overlap of LCA and building simulation. The proposed architecture was pitted against existing LCA tools and only negligible differences were encountered. However, the structure of the simulation will allow for more flexible application and simulation integration in the researched area.

Li et al. implement large scale distributed smart grid co-simulation [Li et al., 2017]. It makes reference of previous frameworks mentioned in other smart grid simulation implementations, such as EPOCH, which is based on HLA. However, it ends up using its own implementation, using Gridlab-D[18], an open-source power system simulation with good third-party integration for the power simulation, and Common Open Research Emulator[19], a node-based distributed server platform. It was concluded that the co-simulation, combining the power grid simulator with a communication network emulator, was able to pass the performance and distribution tests necessary for the algorithms required of the smart grids.

[13] ns-3 is available at https://www.nsnam.org/.
[14] SUMO is available at https://www.eclipse.org/sumo/.
[15] ENVI-met is available at https://www.envi-met.com/.
[16] EPOCHS is available at http://www.cs.cornell.edu/hopkik/epochs.htm.
[17] OpenDSS is available at https://sourceforge.net/projects/electricdss/.
[18] Gridlab-D is available at https://www.gridlabd.org/.
[19] Common Open Research Emulator is available at https://www.nrl.navy.mil/itd/ncs/products/core.

Bragard et al. create a distributed traffic simulation tool, dSUMO, by using the SUMO simulator as a basis [Bragard et al., 2017]. It implements dynamic border management, load balancing and synchronisation. For a given model to work with this tool, it needs to provide an API that's compatible, like SUMO's Traci or VISSIM[20]'s COM interface. The main objective of this work is, however, to use the distributed computing power to speed up the simulation. The platform was able to achieve a speed-up of 5.5 when compared to single-instance SUMO, with proper dynamic load-balancing and decentralised synchronisation.

All of these implementations and their architectural solutions are compiled in Table 3.2. A general overview of the implemented solutions is that, in most cases, an ad hoc solution is chosen. Some occurrences of HLA-based and FMI works exist, whilst a single DDS solution was implemented.

Table 3.2: Architectural Solution by Implementation

	Standard or Architecture					
	DIS	HLA	TENA	FMI	DDS	Ad hoc
[Králíček, 2011]		×				
[Bian et al., 2015]						×
[Schloegl et al., 2015]						×
[Melman et al., 2015]						×
[Cicirelli et al., 2015]		×				
[Brito et al., 2015]		×				
[Albagli et al., 2016]		×				
[Hong et al., 2016]				×		
[Dols et al., 2016]				×		
[Manbachi et al., 2016]						×
[D'Angelo et al., 2016]						×
[Madden and Glaab, 2017]					×	
[Morakinyo et al., 2017]						×
[Garau et al., 2017]						×
[Cellura et al., 2017]						×
[Li et al., 2017]						×
[Bragard et al., 2017]						×

3.3 Disturbances Simulators

The areas that are seen as most promising in which we could make a proof of concept of the co-simulation architecture are wildfires, earthquakes and tsunamis. Parallel to these, atmospheric dispersion is also a very promising and interesting possibility. It relates to many other disturbances, from fires to chemical contamination, in the form of the propagation of particles. Ergo, having a simulator that can accurately simulate how particles from other disturbances propagate is of interest. However, it is important to note that besides for wildfires, input data for the other types of

[20]VISSIM is available at https://www.ptvgroup.com/en/solutions/products/ptv-vissim/.

disturbances are very sparse if not actually non-existent inside FSX. For this reason, the wildfire simulator search was more in-depth than the rest. Nevertheless, the other possible areas were still researched, in the case that a tool that required compatible data showed up.

Some of the possible wildfire simulators to be integrated are analysed ahead.

FlamMap[21] is a fire analysis desktop application that can simulate fire characteristics like fire growth and spread published by the United States Department of Agriculture's Forest Service Fire and Aviation Interagency Incident Applications. It is free to use and runs on Windows. It includes the department's previous tool, FARSITE [Finney, 1998] and various fire behaviour models, such as Rothermel's surface fire spread model [Rothermel, 1972], Van Wagner's crown fire initiation model [Wagner, 1977], Rothermel's crown fire spread model [Rothermel, 1991], Albini's spotting model [Albini, 1979], Finney's or Scott and Reinhardt's crown fire calculation method [Finney, 1998, Scott and Reinhardt, 2001], and Nelson's dead fuel moisture model [Nelson, 2001]. The last one is the only model that allows for conditioning of dead fuels based on the environment's characteristics (such as rain). FlamMap does not support variation in the environment's conditions while the simulation is running. All the parameters are set only through a GUI before the simulation starts, after which one has to wait for the whole run to be finished for the output data.

ForeFire[22] is an open-source set of simulation tools and API designed to perform forest fire simulation, released under the GNU General Public License [Filippi et al., 2014]. It simulates high resolution and large fire fronts of wildland fire. It implements the Rothermel and Balbi surface spread models, and a series of heat flux models [Rothermel, 1972, Balbi et al., 2009]. It is event-based and accepts commands via a command shell, which can open script files. It can progress the time and output data on demand.

Forest Fire Simulation is a simulation tool to be used within CALCHAS, a project which studies the fire conservancy of forests [Kiranoudis, 2013, CALCHAS, 2010]. It is a case study funded by the European Union. Since Forest Fire Simulation's usage outside the CALCHAS project is forbidden, it is not a viable solution for this work. Because of this, it is not considered when comparing the simulators.

Wildfire Analyst[23] is a tool which allows for the offline study of wildfire behaviour. It can calculate common outputs, such as rate of spread, flame length, fireline intensity or crown fire potential. It can generate high definition wind fields based on various weather data. It also can calculate the fire supression capacity, based on criteria defined by the user.

In Table 3.3, a comparison is made between the simulators which are viable for this work. As we can see, FlamMap seems to have more models that can deliver a more detailed simulation, with the spread, crown fire and dead fuel models. Wildfire Analyst can have better information on the flame, rather than the deal fuel models in FlamMap. They require more environment input data for each model. ForeFire can only simulate surface fire spread and heat flux. However, both FlamMap and Wildfire Analyst are complete solutions that have features to analyse and visualise

[21] FlamMap is available at https://www.firelab.org/project/flammap.
[22] ForeFire is available at http://forefire.univ-corse.fr/.
[23] Wildfire Analyst is available at https://www.wildfireanalyst.com/.

the results, while ForeFire only contains the simulator itself. This can be an advantage, seeing that it's more lightweight. FlamMap and Wildfire Analyst can also only be used through their GUI, while ForeFire can be interacted with programmatically. Wildfire Analyst is also a commercial solution.

Table 3.3: Wildfire Simulators Features

	Free	Open-Source	API	Documentation Quality	Features
FlamMap	Yes	No	No	Poor	Fire spread, crown fire, dead fuel
ForeFire	Yes	Yes	Yes	Rich	Fire spread, heat flux
Wildfire Analyst	No	No	Yes	N/A	Fire spread, crown fire, fire intensity

The following are found earthquake simulators that are analysed.

Virtual Quake is an earthquake simulation tool that "performs simulations of fault systems based on stress interactions between fault elements" [Wilson et al., 2017].

OpenSees[24] is a framework that allows the development of apps for simulation of structural systems that are subjected to earthquakes.

In Table 3.4, the earthquake simulators' features are outlined. While Virtual Quake is better oriented to simulate tectonic faults, OpeeSees allows to see how structures are affected by earthquakes.

Table 3.4: Earthquake Simulators Features

	Free	Open-Source	API	Documentation Quality
VirtualQuake	Yes	Yes	Yes	Medium
OpenSees	Yes	No	Yes	Rich

For tsunamis, TOAST (Tsunami Observation And Simulation Terminal)[25] is a commercial tsunami simulation software, which can perform on-the-fly simulation accelerated with graphics processors.

Multipurpose tool Geowave[26] can simulate earthquake, landslide, and volcanic tsunamis using a 4th order Boussinesq equation model. It is published under the GNU General Public License. However, its parameters are not dynamically configurable and are mostly optimised for tsunami simulation. For this reason, it is not included in the earthquake simulator comparison.

The tsunami simulating tools' features are available in Table 3.5. Geowave has a narrower use case, but is non-commercial and has better integration capabilities.

[24] OpenSees is available at https://opensees.berkeley.edu/.
[25] TOAST is available at https://gempa.de/products/toast/.
[26] Geowave is available at http://www.appliedfluids.com/geowave.html.

Table 3.5: Tsunami Simulators Features

	Free	Open-Source	API	Documentation Quality
Geowave	Yes	Yes	No	N/A
TOAST	No	No	No	Rich

HYSPLIT[27] is a tool with a set of atmospheric transport and dispersion models. The models use a hybrid between Lagrangian and Eulerian methods. Some applications include determining the origin of particle emissions and tracking and forecasting the release of particles.

CAMx[28] is a tool with a photochemical grid model which allows for the simulation of air quality, treatment of pollutants and conducting source attribution analysis. It is written in FORTRAN, and available as sources to be built for Unix-based platforms.

GRAL[29] is the Graz Lagrangian Model framework, which allows for the simulation of numerous situations of particle dispersion. The model's original purpose was to tackle low wind speed conditions, as well as dispersion of emissions from road tunnel portals.

openair[30] is an R module for the analysis of air quality. Its features include access to data from air pollution monitoring sites, air and pollution roses, and access to pre-calculated HYSPLIT annual back trajectories.

Their features are outlined in the Table 3.6. openair, unlike HYSPLIT, CAMx and GRAL, is a module which is more useful for analysing air data, rather than simulating it. For the rest, they are programs with overall feature parity.

Table 3.6: Atmospheric Dispersion Simulators Features

	Free	Open-Source	API	Documentation Quality
HYSPLIT	Yes	No	No	Medium
CAMx	Yes	Yes	No	Rich
GRAL	Yes	Yes	No	Rich
openair	Yes	Yes	No	Rich

3.4 Summary

This chapter presented us with the literature trends and information regarding co-simulation architectures and standards. Important information in how the different architectures might be used was found. Examples of such architectures, standards and protocols are DIS, HLA, FMI, DDS and ad hoc implementations. An ad hoc implementation would be, essentially, the lack of use of a given standard, framework, or protocol. This is, however, a common occurrence, as one is able to see in Table 3.2.

[27] HYSPLIT is available at https://www.ready.noaa.gov/HYSPLIT.php.
[28] CAMx is available at http://www.camx.com/home.aspx.
[29] GRAL is available at http://lampz.tugraz.at/~gral/.
[30] openair is available at https://davidcarslaw.github.io/openair/.

The observable trend in current co-simulation architectures is that, as of late, they are using their own solutions with an ad hoc implementation, and not exactly following an available existing co-simulation standard. However, there are some older examples of implementations using HLA and FMI. Another special case was the proposal of DDS, a data transmission protocol as a replacement for HLA.

The shift into custom ad hoc solutions might be a consequence of the simulation models, especially in energy, being available in tools with their own or no coupling solutions, and the lack of necessity of using standards which add unneeded features in a given architecture.

As for the simulators, models for earthquakes, tsunamis and air dispersion require a lot of input data and non-trivial parameter adjustments. As such, the fire simulators are of more interest in this work. Between those, FlamMap and ForeFire are possible solutions, depending on the available input data.

Chapter 4

Proposed Solution

In this chapter, there will be an overview of the projected solution's architecture, along with its technological choices. Afterwards, the work plan and risk analysis are presented.

4.1 Projected Architecture

The new projected architecture could vary with regards to the concurrent work that has been developed in The Platform. Esteves worked on bringing distributed simulation to The Platform, in order to achieve better performance and a greater virtual region of simulation [Esteves, 2020]. Costa developed more efficient and secure communication middleware, to replace AgentService's slow and insecure message distribution [Costa, 2020]. Therefore, there was a need for a coordinated effort to choose a common architecture for all of the ongoing developments.

If a simulation-oriented standard or protocol architecture were to be chosen, then an HLA RTI would be the most likely contender, seeing that it's the most used architecture in the literature. However, properly using an HLA RTI would imply big shifts in the entire Platform's architecture. Some management aspects of HLA might not have a feasible way of being integrated with FSX and the existing components of The Platform and might actually overlap with existing concepts defined within the project and FSX.

If such a standard or protocol did not provide any seemingly important advantage versus using a common architecture for the Disturbances Manager and its co-simulators, and the other ongoing projects, The Platform would probably have an architecture which is similar to the one proposed by Almeida [Almeida, 2017]. A diagram of The Platform's components with such a choice can be observed in Fig. 4.1, where the connection between the co-simulators, the Disturbances Manager and new instances of the Simulator communicate using the chosen standard or protocol.

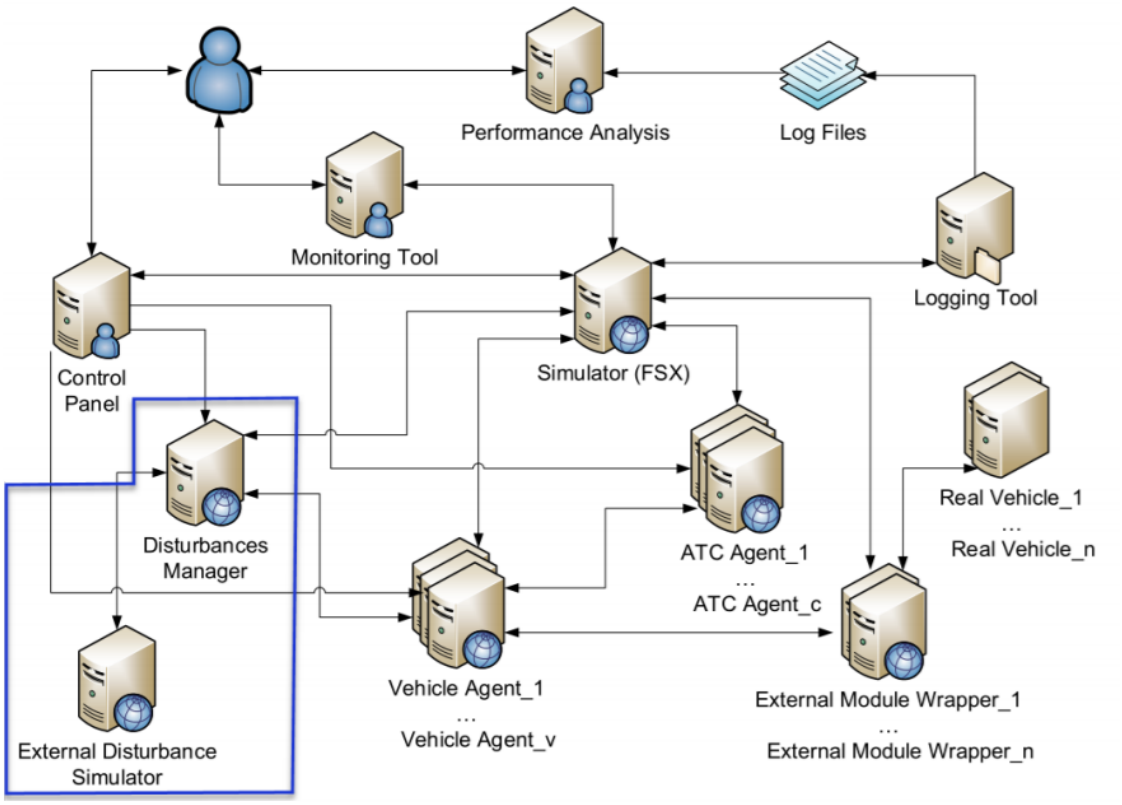


Figure 4.1: The Platform's Proposed Architecture [Almeida, 2017]

4.2 Technological Choices

As observed in the trends from section 3.2, there aren't many implementations following distributed and co-simulation standards. It is unclear if they were unsuitable for their needs or if there were simply not known (although some cases mentioned frameworks that used them but chose other implementations anyway). Because these standards feel appropriate to ensure that the Disturbances Manager has a proper future-proof solution to coupling new co-simulators, it seems pertinent to use a standard in the architecture. As mentioned before, the overall best choice, in this case, would be to use HLA, using a publicly available HLA RTI implementation. This choice would suffer, however, of some issues that were mentioned in the previous section.

The pros and cons of both HLA and other protocols (as is the case of DDS) in the context of The Platform were discussed and carefully analysed within the current developing group, in order to try and achieve a common architectural ground. The main point was that if the advantages of choosing the same architecture aligned with both of these other works' objectives and work plan, then a common architecture, allowing distributed and co-simulation architecture, and a secure and efficient communication service, would be chosen. If not, each of the works would have the architecture that is best suited for their purpose.

4.2.1 Co-Simulation Architecture

Initially, the outcome of the coordination was to make a first effort communication test using DDS, for the projects trying to achieve distributed and co-simulation. Advantages such as hot-plugging and overall implementation simplicity that didn't imply a major architectural (which would be the case for HLA) change were seen as positive for choosing this protocol. It also supported various programming languages. Several DDS implementations exist, but only non-commercial solutions are considered for this project. The following are available free DDS implementations: Vortex OpenSplice Community Edition[1], Eclipse Cyclone DDS[2] and OpenDDS[3]. OpenDDS only has bindings for C++ and Java, thus not being a viable option for this project (because of the programming language used in The Platform, C#). As for Eclipse Cyclone DDS, early releases were experimental in nature, with C compatibility. At the time of writing (with the latest version being 0.6.0 Florestan), only bindings for C++ had been implemented and security is scheduled for a future release. Given this, Eclipse Cyclone DDS is also not a viable option. Taking lead from the existing experiments and the availability of DDS implementations, Vortex OpenSplice Community Edition was chosen [Madden and Glaab, 2017]. It has bindings for C# and Python, and it has security via the DDS Secure standard [OMG, 2018]. However, an increasing number of issues occurred. It was found that this implementation did not behave similarly between different programming languages' clients, having different behaviours between its Python and C# client in the way that the data was consumed from the topics. Whilst things worked well between clients of the same language, seemingly equivalent basic Hello World test projects in the two languages would not work when interacting between themselves. Messages would get sent repeatedly and, in some instances, could even cause the receiving client to crash. The clients do not have a uniform interface, meaning that to define the same behaviour in the different programming languages is done in a vastly different way. Another aspect is that these clients were actually only a wrapper for a service running in the background, which had to be previously installed in each machine. Furthermore, this service had unneeded functionality in the form of extensions, making it bloated, and established connections mainly using peer discovery, which besides not working outside of a subnet, did not always work even within one. Finally, by looking at the provided support to the community, it was found that some of these issues existed for other users and were fixed in the commercial version of this product. These complications discouraged the use of OpenSplice and thus, the use of DDS was abandoned, as there was no other free DDS implementation with the requirements that were needed. Creating our own DDS standard implementation is out of the question for two different reasons. Firstly, the standards are extensive and would take more time to implement than is available for the completion of this work. A simple reference for this point is the Eclipse Cyclone DDS project, which even with the help of a company that has been working on DDS for some time still has its development undergoing. Secondly, not all features of the

[1] Vortex OpenSplice Community Edition is available at https://github.com/ADLINK-IST/opensplice.
[2] Eclipse Cyclone DDS is available at https://projects.eclipse.org/projects/iot.cyclonedds.
[3] OpenDDS is available at https://opendds.org/.

standards would be required and, because of that, there would be no reason to fully implement the specification, as doing that would be lost time.

4.2.2 Middleware Selection

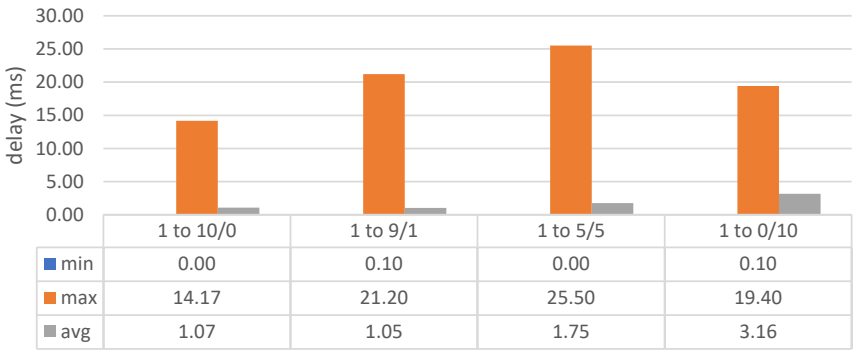
From a pre-selection of message queue implementations that would support a variety of programming languages, tests were run on ActiveMQ[4], a Java broker which implements the Java Message Service (JMS) and other message queue protocols, such as Advanced Message Queuing Protocol (AMQP), Simple (or Streaming) Text Oriented Message Protocol (STOMP), Message Queuing Telemetry Transport (MQTT), etc., and RabbitMQ[5], an Erlang broker which originally implemented AMQP, but has since added support for MQTT, STOMP and others. JMS is part of the Java Enterprise Edition.

To find out how good the compatibility between different programming language's clients is, a message would be broadcast from a C# client, The Platform's main programming language, and other prospects to be used in the co-simulation integration, like Java and Python. The time for the message to be received since it was sent was measured. The C# producer would instantly send 1000 messages to 10 consumers: first, 10 C# clients, then 9 C# clients and a single Java or Python client (to represent the probable use case where there's mostly The Platform's components communicating), then half of each, then finally only Java or Python clients. The broker's native protocols and suggested clients for each language were used. However, because the JMS is not part of Java Standard Edition, the tests weren't performed on ActiveMQ Java consumers. The tests were performed three times to diminish the impact of outliers.

The results can be seen in Figs. 4.2, 4.3 and 4.4, with times expressed in milliseconds. For ActiveMQ with C# and Python clients, as we increase the number of Python clients, the average time increases and tops out at 3.1553 ms. The maximum value can go as high as 25.5 ms. For RabbitMQ with C# and Python clients, the same pattern is found, but this time at a much worse scale. The average time goes up to 67 ms, while the maximum hits 117.4 ms when only using Python consumers. For RabbitMQ with C# and Java clients, the results are on the same scale as the previous test. The times for RabbitMQ consumers to receive a message go exceedingly high to a maximum of 139.3 ms.

This first batch of tests shows that RabbitMQ performs much worse when compared to ActiveMQ, even when only using C# producers and consumers. The latter only worsens when decreasing the number of C# clients. On the other hand, it was noticed that ActiveMQ had a really big memory usage, in the order of the gigabytes, while RabbitMQ kept in the order of megabytes. However, it was noticed that this test might not be representative of the expected behaviour of The Platform, and is akin to a stress test. Because of this, a new "slow" test was devised, where the producer would wait 1 millisecond between broadcasting each message.

[4]ActiveMQ is available at `http://activemq.apache.org/`.
[5]RabbitMQ is available at `https://www.rabbitmq.com/`.

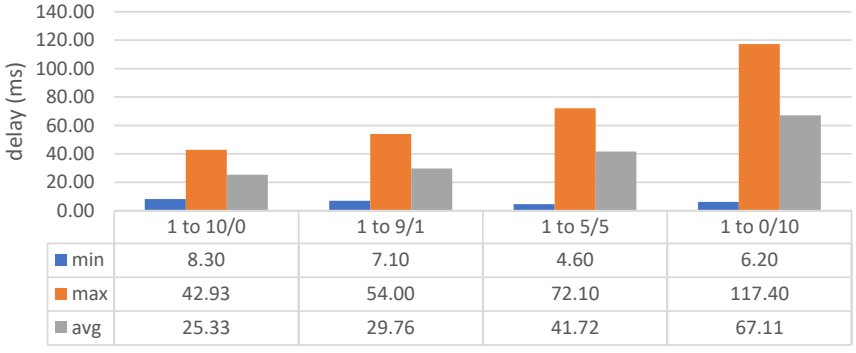


	1 to 10/0	1 to 9/1	1 to 5/5	1 to 0/10
min	0.00	0.10	0.00	0.10
max	14.17	21.20	25.50	19.40
avg	1.07	1.05	1.75	3.16

Figure 4.2: Stress Test of ActiveMQ with C# and Python Clients

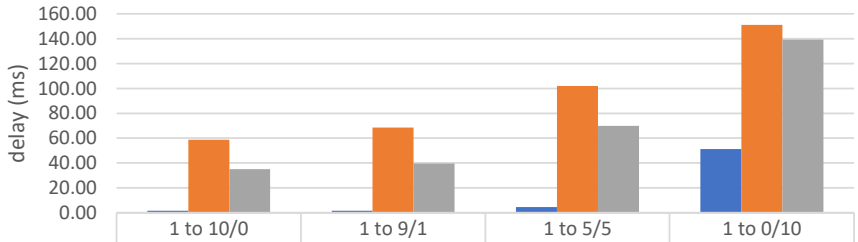
The results can be seen in Figs. 4.5, 4.6 and 4.7, with times expressed in milliseconds. For ActiveMQ with C# and Python clients, it keeps relatively the same profile as the stress test. However, for RabbitMQ with C# and Python clients, the average time never exceeds 2 ms, and even the maximum delay is lower, increasing the case of only Python clients, but with times lower than the ActiveMQ counterpart. For RabbitMQ with C# and Java clients, the average is always close to zero, with the maximum not exceeding 25 ms.

After this test, it is assumed that RabbitMQ is better in scenarios similar to what is existing in The Platform, where no such high number of messages is sent at the exact same time. Nonetheless, with the increase of disturbances readings and agents, the numbers can get to a point similar to the ones in the stress test. However, other challenges would have to be tackled beforehand, such as the number of agents an FSX instance supports. Other measures could also be applied to diminish the impact, such as having more instances of the broker. This is, however, out of the scope of this work and better analysed in the works of Costa and Esteves [Costa, 2020, Esteves, 2020]. In alignment with other tests performed in those works regarding security and scaling, RabbitMQ was found to be the better choice.



	1 to 10/0	1 to 9/1	1 to 5/5	1 to 0/10
min	8.30	7.10	4.60	6.20
max	42.93	54.00	72.10	117.40
avg	25.33	29.76	41.72	67.11

Figure 4.3: Stress Test of RabbitMQ with C# and Python Clients

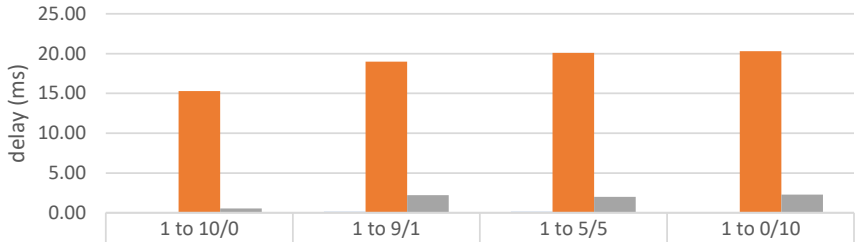


	1 to 10/0	1 to 9/1	1 to 5/5	1 to 0/10
min	1.67	1.70	4.57	51.20
max	58.83	68.43	101.97	151.00
avg	35.12	39.56	69.99	139.31

Figure 4.4: Stress Test of RabbitMQ with C# and Java Clients

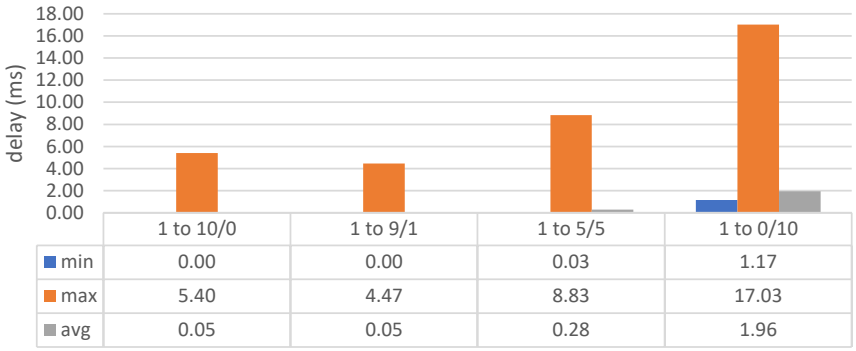
4.2.3 External Simulator

For the simulator to be coupled, ForeFire will be used. Unlike most other simulators listed, which work on an input parameters, output data basis, with no way to programmatically achieve this, ForeFire has an interactive command shell in which we can dynamically specify simulation parameters and time jumps. Having the possibility to generate on-demand and on the fly output was a key feature that made this simulator stand out from all of the others. The inclusion of Python tools to help generate the simulation data was also helpful. Because of this, the middleware to link ForeFire to the Disturbances Manager is made in Python. C# is not the best solution since ForeFire runs in Unix-based environments. The choice between Python and Java is made based on the ease of using the pre-distributed Python tools in ForeFire. Finally, the data used in fire simulations are easier to obtain from FSX than other, more complicated data used in earthquake or tsunami simulation. For example, ForeFire requires fuel, elevation and wind data for a basic simulation, which FSX contains, unlike tectonic, atmospheric particles or wave data. One thing to take into account is that only the surface fire spread model of ForeFire will be used, so no advantage would be had by choosing a simulator with more models, like FARSITE, because the same Rothermel spread model would be chosen. This also means that there will be no information to be had on



	1 to 10/0	1 to 9/1	1 to 5/5	1 to 0/10
min	0.00	0.10	0.10	0.00
max	15.30	19.00	20.10	20.30
avg	0.54	2.20	2.00	2.30

Figure 4.5: Slow Test of ActiveMQ with C# and Python Clients



	1 to 10/0	1 to 9/1	1 to 5/5	1 to 0/10
min	0.00	0.00	0.03	1.17
max	5.40	4.47	8.83	17.03
avg	0.05	0.05	0.28	1.96

Figure 4.6: Slow Test of RabbitMQ with C# and Python Clients

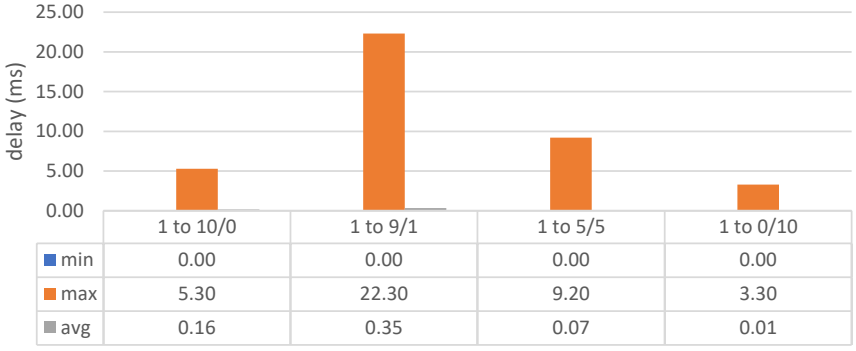
how much the fire is burning or if the fuel is depleted, as the model does not have those features.

4.3 Planning

To implement the proposed solution, the following implementation steps were followed:

- Analysis and preparation of the input data required by the external simulator – it is necessary to understand how ForeFire needs its input data to be provided, and where this data can come from. Having correctly identified this input format and where to retrieve the input data from, tools to convert the data to be used by ForeFire will be developed;

- Creation of a communication middleware between the external simulator and the Disturbances Manager – using Python, a middleware to couple ForeFire to the Disturbances Manager will be implemented. With this, a communication protocol between the two will also be established;



	1 to 10/0	1 to 9/1	1 to 5/5	1 to 0/10
min	0.00	0.00	0.00	0.00
max	5.30	22.30	9.20	3.30
avg	0.16	0.35	0.07	0.01

Figure 4.7: Slow Test of RabbitMQ with C# and Java Clients

- Implementation of the co-simulation architecture and externally simulated disturbances – After having the communication middleware and protocol, implementing the protocol on the Disturbances Manager is required;
- Test and validation – guarantee that the coupling is performing correctly, evaluating the solution's functionality and the overall performance of the co-simulation architecture.

Chapter 5

Implementation

In this chapter, the detailing of the implementation and the considerations that influenced it are described.

5.1 Input Data

This section will describe the implementation steps in order to achieve the wildfire simulation in accordance with FSX's data. In short, four types of data are extracted from FSX's files and run-time. Land class and elevation data are extracted from the BGL files and kept raw in various files for easy access by the Python middleware. Using various BGL files, a list of airports is compiled using MakeRunways and kept on a CSV file. Finally, the weather information is requested for the necessary airports. After having access to all of this data, the Python middleware manipulates it to the format required by ForeFire, which then receives the layers of fuel indexes, elevation and wind components. The fuels are pre-described in a different file. Figure 5.1 shows the overall structure of the information flow for all required data for ForeFire to perform its wildfire spread simulation. One thing to notice is that the figure only shows the information flow and does not take into account the communication between the different components that was required to achieve it.

5.1.1 BGL File Format

The BGL File Format is a type of file used to store information related to simulators which use Microsoft ESP [FSDeveloper, 2020]. It is a binary file with a common base structure. However, for different types of information and simulators, the data is stored in different ways.

As can be seen in Fig. 5.2, the BGL file has 3 main records: the header, sections, and sub-sections. They contain metadata about the contents of each file. Each of these records states the geographical region that they refer to, via quad mesh identifiers (QMID), their size, and the offset to the start of the relevant record, e.g the first section offset, first subsection offset, first data record offset. The file always starts with a header which contains the number of sections the file has. The

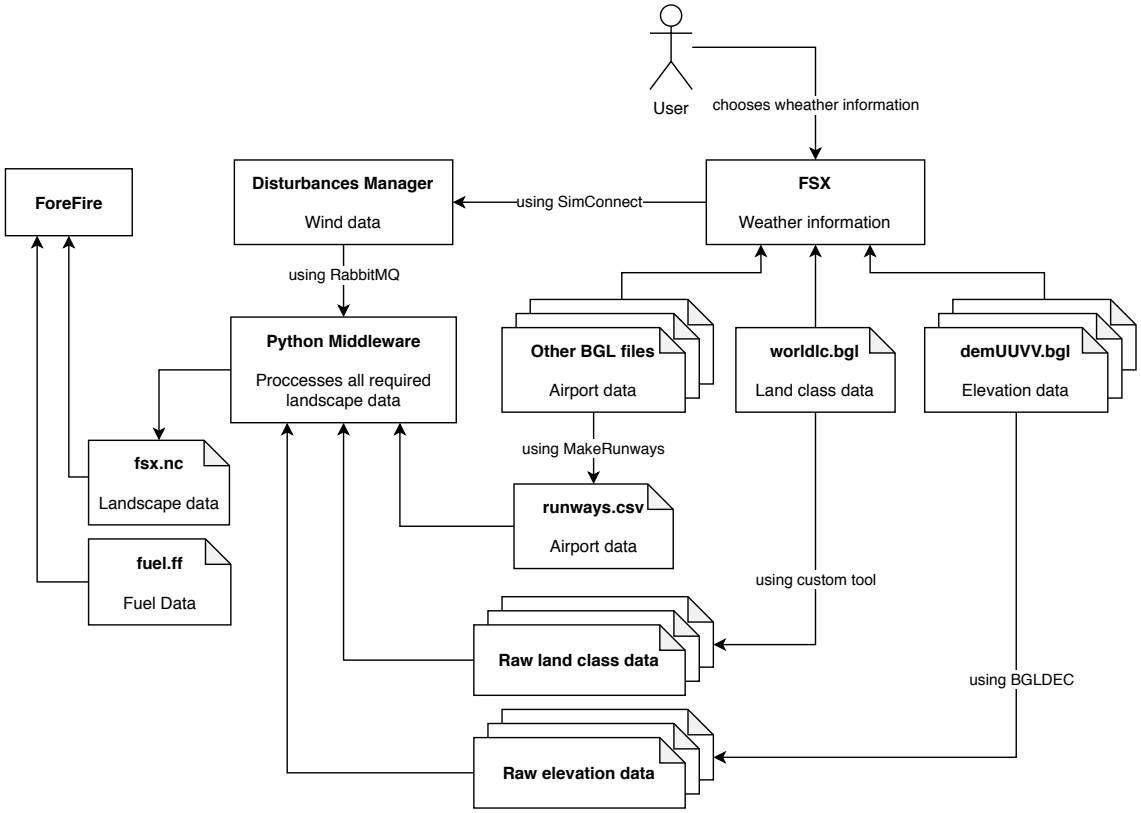


Figure 5.1: Information Flow

sections, in turn, contain the number of subsections each section has, and information about the type of data stored in them. For this work, only sections of type *TerrainElevation* and *TerrainLandClass* are relevant. Each subsection contains the number of data records existing within.

The information regarding land classification and elevation, which is relevant later on in this work, is stored within TRQ1 records. These records have headers which indicate the types of the parent sections, with them being the aforementioned ones. They also contain the compression scheme of their data. Compression algorithms used in the land class data are Ziv and Lempel's LZ1 [Ziv and Lempel, 1977], delta encoding, and bit packing, whereas elevation data also use Microsoft's proprietary progressive transform coder (PTC) compression [Malvar, 2000]. They also include month indicators for seasonal sensitive data, which is, in this case, left empty. The final, decompressed data is a 257 by 257 grid of integers. They are 8 bit in the case of land class data, and 16 bit in the case of elevation data. These grids overlap on the edges with the adjacent grid, i.e. the edges of adjacent grids are the same. This means that when stitching these, the final edge size for a grid will be $256n+1$ (hence the grid size of $256+1$), where n is the number of stitched grids in that edge's orientation.

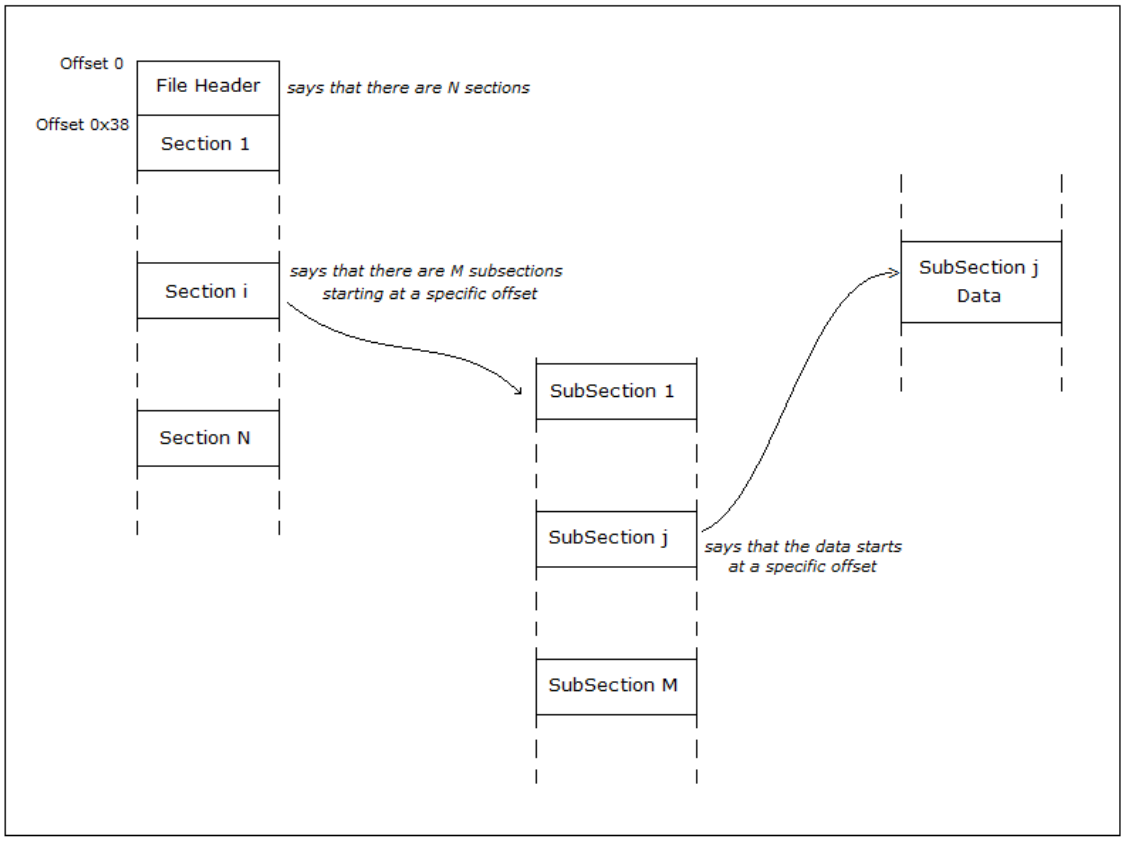


Figure 5.2: BGL File Format's General Structure [FSDeveloper, 2020]

5.1.2 QMID

QMID is the way that stored simulation data's location is indicated in some flight simulators [Microsoft Corporation, 2008c]. They refer to the size of the grid used in a data record. QMID is consisted of three values: l, u and v. l represents the level of the grid, starting at 2, which divides each hemisphere into three regions. Increasing the value of l by one divides each previous grid into four. u and v refer to the number of the grid at a given level, for longitude and latitude, respectively. This can be seen in Fig. 5.3.

5.1.3 ForeFire Input Format

ForeFire is executed by running the *CommandShell* binary, a shell interface which accepts commands to define certain parameters. However, some of the necessary data to run a simulation is defined in a file. The information about the types of existing fuels is loaded via the *fuelsTableFile* parameter, whilst the information about the fuel, elevation and wind maps is loaded using the *loadData* command.

The fuel file is essentially a comma-separated values (CSV) file, with predefined fuels and their parameters in accordance with the fuel model. The indexes refer to the CORINE Land Cover

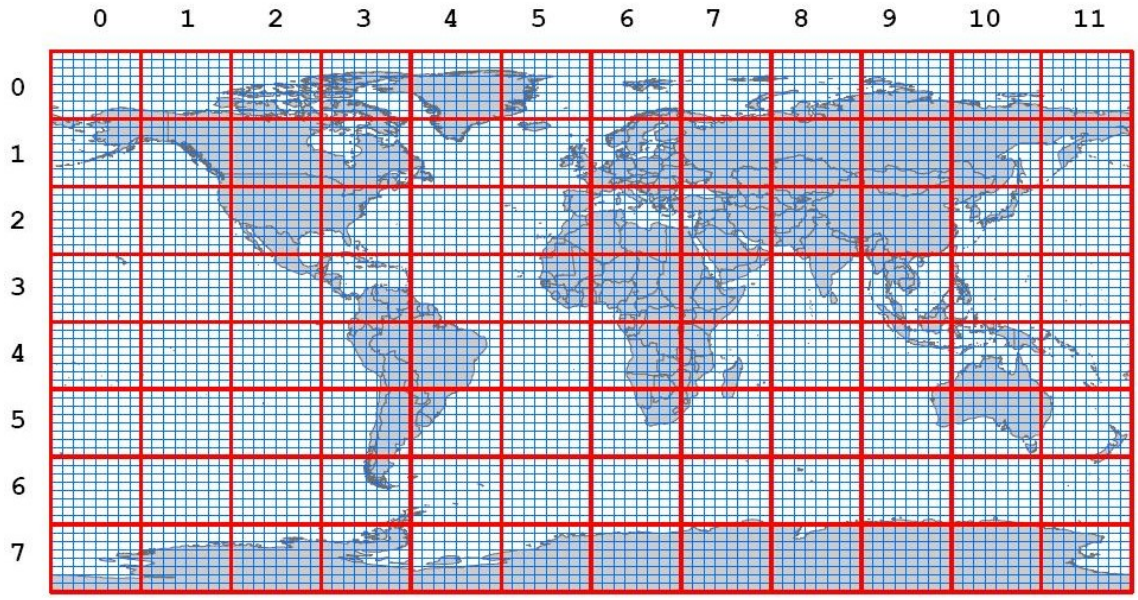


Figure 5.3: FSX's World QMID Grids for Levels 4 and 7 (adapted from [Microsoft Corporation, 2008c])

(CLC), a European database of biophysical occupation of land [Bossard et al., 2000]. This file is included by default with the simulator and has not been changed.

The fuel, elevation and wind maps are stored in a Network Common Data Form (netCDF) file. This is a binary file format for exchanging scientific data. However, there's a Common Data Language (CDL) format which is a readable format of a netCDF file. Using tools such as *ncdump* and *ncgen*, the CDL file can be generated from a netCDF one and vice-versa. For ForeFire, this is an example of the CDL format:

```
1  netcdf fsx {
2  dimensions:
3          ...
4  variables:
5          int fuel(ft, fz, fy, fx) ;
6                  fuel:type = "fuel" ;
7          double altitude(at, az, ay, ax) ;
8                  altitude:type = "data" ;
9          double windU(wt, wz, wy, wx) ;
10                 windU:type = "data" ;
11         double windV(wt, wz, wy, wx) ;
12                 windV:type = "data" ;
13         char domain ;
14                 domain:type = "domain" ;
15                 domain:SWx = -9601306.f ;
16                 domain:SWy = 4775530.f ;
17                 domain:SWz = 0 ;
```

```
18                 domain:Lx = 417448.1f ;
19                 domain:Ly = 413577.2f ;
20                 domain:Lz = 0 ;
21                 domain:t0 = 0 ;
22                 domain:Lt = Infinityf ;
23         char parameters ;
24                 parameters:type = "parameters" ;
25                 parameters:projection = "EPSG:3395" ;
26 ...
27 data:
28  fuel = ...
29  altitude = ...
30  windU = ...
31  windV = ...
32 }
```

Listing 5.1: ForeFire netCDF format

The input format for fuel is an array of integers, indexes which assign a fuel from the fuel file to a tile in the fuel grid.

The input format for elevation and wind maps are arrays with decimal values entries. Items in the elevation array correspond to the elevation in metres. For the wind maps, we have two arrays, where one indicates the zonal flow, corresponding to the x-axis, and the other indicating the meridional flow, which corresponds to the y-axis. These wind components are expressed in metres per second.

The domain field defines the boundaries of the simulated area with the coordinates of the south-westernmost corner of the region to be considered, and its length and height. These coordinates, length and height should all be in the map projection selected in the parameters.

5.1.4 Fuel Data

FSX contains a lot of terrain-related information, e.g. elevation, land and water classification, etc. For the fuel data, land classification is the most adequate data to create a fuel map, seeing that even ForeFire itself uses essentially the same kind of information as fuel types.

However, ForeFire uses the previously mentioned CLC, while FSX uses a modified Olson Land Classification [Olson et al., 2001]. Consequently, there's a need to match the FSX's data with the CLC. Since the simulation area for the fire is dynamically chosen, it is necessary to map all of the Olson Land classification entries to CLC. This was done by choosing the closest or most relevant CLC entry for a given classification, and with the help of CLC nomenclature guidelines [Kosztra et al., 2019]. The mapping can be observed in Appendix A.

The land classification data for FSX is stored in a file called *worldlc.bgl*. This file is a BGL file, structured as described in section 5.1.1.

To extract the data from this file, a tool which is able to correctly parse the entries in a BGL file and decompress them was developed. As was mentioned in the aforementioned section, the land

class file only uses LZ1, Delta, and Bitpack compression algorithms, which have the corresponding decompression algorithms described in public BGL file format documentation [FSDeveloper, 2020].

The final, raw output is stored in a file for each level 7 QMID grid, to easily access the necessary land classification data later on.

5.1.5 Elevation Data

FSX's elevation data is stored in various files in the *Scenery* directory. In this directory, there are various folders with a four-digit name. These digits represent QMID u and v values for a level 4 grid, as can be seen in Fig. 5.3. Each of these folders will then have a file named *demUUVV.bgl*, where UU and VV represent the QMID u and v values.

There is a particularity in these BGL files: the compression algorithm used is PTC. This algorithm is not described anywhere publicly, with a lot of common tools to navigate BGL files not having support for decompressing data that was compressed using this algorithm. However, there exists a tool, called BGLDEC[1], which can decompress files using the PTC algorithm.

Because the land classification data is only available as QMID level 7 grids, the same level is used for the elevation data. With the help of a Python script, BGLDEC was used to decompress all of the elevation files and then store only the relevant raw level data, deleting higher or lower resolution ones.

5.1.6 Wind Maps

In FSX, the wind information is not stored in any file. It is, instead, described in run-time, before starting a flight, along with other weather information. One can choose from certain existing presets or manually describe one. An option to use real-world data exists but isn't functional, as the service from where the simulator obtained the data is no longer available. Besides manually setting the weather information, there are programs which inject the weather into the game using the SimConnect SDK. Tools that can perform this task are, for example, Active Sky[2] and FSrealWX[3]. However, these tools are payware and therefore were not considered for this task.

The weather information is stored for various weather stations that FSX has scattered around the globe. These stations correspond to existing airports and are denoted using the airport's International Civil Aviation Organization (ICAO) code. ICAO codes consist of four letters, with the first two letters indicating region and country, respectively. However, not all airports have corresponding weather stations. FSX includes many minor airports and airfields, which often don't have weather stations nor an ICAO code, which is usually reserved for larger aerodromes. Figure 5.4 shows an example of the weather stations available in a given area. The information is stored in the Meteorological Aerodrome Report (METAR) format, a human-readable format used

[1] BGLDEC is available at https://www.fsdeveloper.com/forum/threads/bgldec-a-resample-bgl-decompressor.433789/.

[2] Active Sky is available at https://hifisimtech.com/.

[3] FSrealWX is available at https://www.fsrealwx.de/.

in aviation to report weather information. A relevant detail of this format is that wind is described with the direction the wind is blowing from, in degrees, and its speed, in knots. When FSX needs to have the weather information for arbitrary coordinates, it generates an interpolated METAR observation for the location.

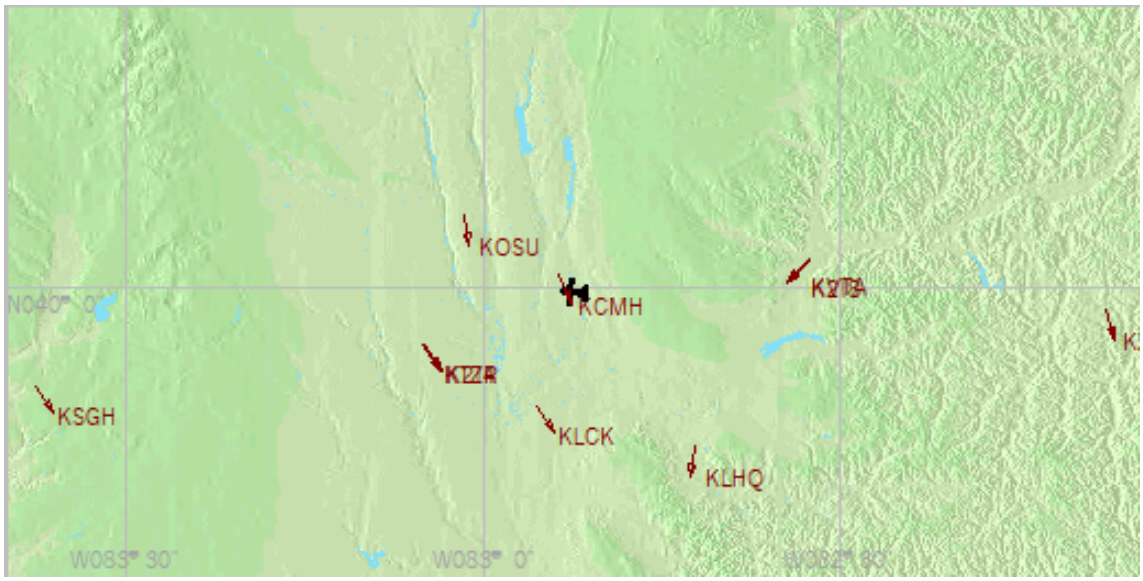


Figure 5.4: FSX's Weather Stations and Wind Information. The airplane denotes the spawn point for the player's airplane. In brown, there's the ICAO codes for available weather stations and, next to them, an arrow with the wind direction reported there.

SimConnect's SDK has three relevant functions for this purpose:

- *SimConnect_WeatherRequestInterpolatedObservation* – This function takes the coordinates of a location, in latitude, longitude and altitude, and returns an interpolated METAR report for that location.
- *SimConnect_WeatherRequestObservationAtNearestStation* – This function takes the coordinates of a location, in latitude, longitude and altitude, and returns a METAR report for the location's closest weather station.
- *SimConnect_WeatherRequestObservationAtStation* – This function takes the ICAO code of a weather station and returns its METAR report.

Hence, an initial approach to extracting the wind information was to request interpolated observations for coordinates in the centre of the tiles corresponding to the grids used in the land class and elevation data. However, SimConnect's SDK was found to have an extremely low throughput, yielding anywhere from 40 to 60 requests per second. On a worst-case scenario, a 257 by 257 would require 66049 readings, which in turn would take around half an hour to finish. Because the weather can be arbitrarily changed, SimConnect requests for FSX to generate interpolated readings for a given area are not a one-shot task. Because The Platform is used to perform several

tests on vehicular missions, having such a long pre-processing time, which is often comparable to a test's duration, is unacceptable.

Because of the time limitation of trying to request the interpolated readings, the alternative approach was to instead create a list of all the available airports in FSX, which would be equivalent to enumerating all the existing weather stations, filter them by airports inside the relevant area, request the METAR reports for those, and finally, interpolate them for all the tiles for the required grid.

Make Runways[4] was used to create a CSV file of all existing FSX airports. This file includes various fields, but the necessary ones are the airport's code and its longitude and latitude. As mentioned before, these include many minor airports and airfields, some of which don't have IACO codes (and instead have an International Air Transport Association code or a pseudo-ICAO code). For that reason, airports are filtered by having a valid ICAO code and then by a given bounding box. Finally, the METAR reports for each of these airports is requested. Invalid airports, which don't have either a valid ICAO or an associated weather station, in which case SimConnect will return an exception upon METAR request, are permanently deleted from the airports' file.

Before interpolating the values, which will be done directly on their zonal and meridional components, the METAR information is parsed and the direction and speed values of the wind are extracted. Firstly, the magnitude of the speed is converted to metres per second. Finally, to convert them to the necessary u and v components, we can use $u = s \times cos(90 - d + 180)$ and $v = s \times sin(90 - d + 180)$ where s is the speed in metres per second and d is the direction the wind is blowing from in degrees. This is because the wind direction is given in the angles of a compass, with 0 degrees at north, 90 degrees at east, 180 degrees at south and 270 degrees at west, unlike the usual mathematical notation. For this reason, d is subtracted from 90 degrees to start at north and increase the angle clockwise. Also, 180 degrees are added to get the opposite direction, considering that the d is the direction where the wind is blowing from, and not where it is blowing to. After applying geometrical transformation, the result is then $u = -s \times sin(d)$ and $v = -s \times cos(d)$ where simply reversing and negating the typical sine and cosine components for the x and y axes allows for the correct calculation of the zonal and meridional components.

Searching the literature, it is possible to find comparisons between interpolation algorithms for wind [Luo et al., 2008]. Different interpolation methods are usually used, separated by deterministic and geostatistical methods (the latter are stochastic). The geostatistical methods are more complex models to implement and require some different wind models to exist. As for deterministic methods, out of all the compared ones, the Inverse Distance Weighting method was not only the simplest to implement, but was the only one not to present unnaturally smoothed values for the interpolation. This method gives more weight to points closest to the interpolated value. A version of this method was implemented, and two different weightings were tested.

Algorithm 1 shows how the interpolation for the converted heading/speed to zonal and meridional components is performed. The interpolation uses the two closest airports to a given location in the centre of a tile for the chosen grid. A linear interpolation was thought of at first. Consider

[4]Make Runways is available at `http://fsuipc.simflight.com/beta/MakeRwys.zip`.

Algorithm 1 Wind Components Interpolation

Require: $left$, $right$, top, $bottom$, u_{max}, u_{min}, v_{max}, v_{min}, $airports$, $winds$
  $wind_u \leftarrow$ new list
  $wind_v \leftarrow$ new list
  $y \leftarrow (v_{max} - v_{min} + 1) * 256 + 1$
  $x \leftarrow (u_{max} - u_{min} + 1) * 256 + 1$
  for $j = 0$ **to** y **do**
    for $i = 0$ **to** x **do**
      $lon \leftarrow left + (right - left)/x * i + 0.5 * (right - left)$
      $lat \leftarrow top - (top - bottom)/y * j - 0.5 * (top - bottom)$
      $distances \leftarrow$ new list
      for all $airports$ **do**
        $distances$.append($(lat - airport_{lat})^2 + (lon - airport_{lon})^2$)
      end for
      $closest\_airports \leftarrow distances$.indirect_sort()[:2]
      $total\_distance \leftarrow distances[closest\_airports[0]] + distances[closest\_airports[1]]$
      $u = distances[closest\_airports[0]]/total\_distance * winds[closest\_airports[1]][0] + distances[closest\_airports[1]]/total\_distance * winds[closest\_airports[0]][0]$
      $v = distances[closest\_airports[0]]/total\_distance * winds[closest\_airports[1]][1] + distances[closest\_airports[1]]/total\_distance * winds[closest\_airports[0]][1]$
      $wind_u$.append(u)
      $wind_v$.append(v)
    end for
  end for
  return $wind_u$, $wind_v$

two airports A and B, and the distances from a certain location to these airports a and b, respectively. To calculate the coefficient for the wind values of airport B, one would use the expression

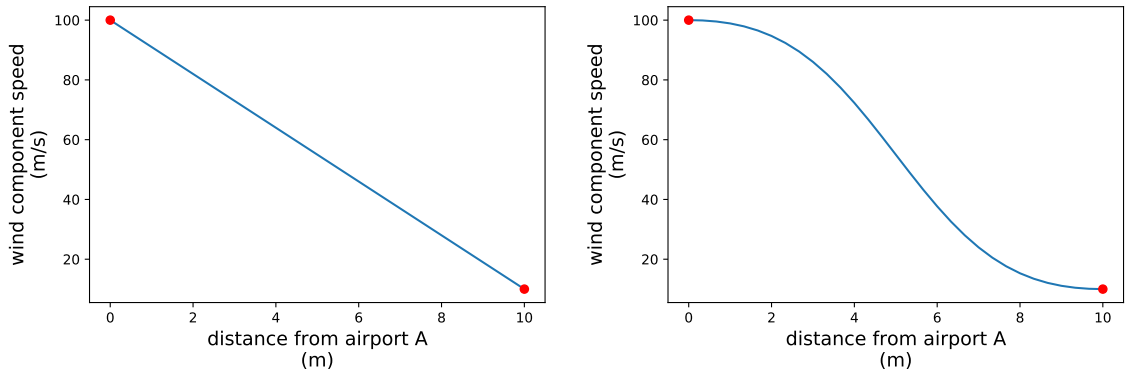
$$\frac{a}{a+b}$$

seeing that when the distance to airport A increases, the more weight should be given to the wind of airport B. Therefore, a linear wind component interpolation would look like

$$wind_L = \frac{b}{a+b} * wind_A + \frac{a}{a+b} * wind_B$$

for a given location L. For a case where the location is colinear with the two airports and the distance between the airports A and B is 10 metres, meaning that $a+b=10$, and for wind component values of $wind_A = 100$ and $wind_B = 10$, the interpolation results can be seen in Fig. 5.5a.

However, a linear interpolation would cause two problems. The first is that to calculate these distances, we would need the square roots of the squares of the distance differences. The square root is a computationally intensive operation when compared to other basic algebraic operations. The second is that when there are vastly different wind values for the two closest airports, just a little distance from any airport will produce a fairly unnatural difference where the other airport's



(a) Linear

(b) Quadratic

Figure 5.5: Wind Interpolation Examples

wind value is taken too much into account. When near to an airport, that airport's wind value should have a stronger weight than the second closest airport's.

To prevent this, to calculate the calculate the coefficient for the wind values of airport B one can instead use

$$\frac{a^2}{a^2+b^2}$$

where there's no need for square root operations. For this quadratic-based interpolation, a wind component is calculated using

$$wind_L = \frac{b^2}{a^2+b^2} * wind_A + \frac{a^2}{a^2+b^2} * wind_B$$

for a given location L. The variation on the wind component values using the same example as before can be seen in Fig. 5.5b. The main difference from the linear interpolation is that when near an airport, the values decrease and increase in a smoother way. This is because even though the distance between airports is constant, the denominator on the coefficient now varies and is higher when closer to an airport than when in the middle, whereas in the linear interpolation the denominator would be constant.

Consequently, quadratic interpolation is the one being currently used. However, this interpolation is naive. In no way does it take into account actual wind behaviour. An example of this is the case where currents with the same speed flowing in opposite directions meet. It might be that this interpolation would produce a low or non-existent wind speed where instead there would be turbulence and high wind speeds. It should also not be seen as a proper replica of FSX's own interpolation, which runs only for existing aircraft and not thousands of points. It is, however, a computationally cheap approximation which is enough for approximating winds in a given area.

5.2 Co-Simulation Architecture

To execute ForeFire, using the *CommandShell* binary, a Python middleware was developed. It communicates with the binary using the Pexpect[5] Python module. It uses Unix pseudoterminals to communicate with applications via their standard input and output. Because ForeFire runs in a Unix environment and The Platform and FSX run in Windows environments, ForeFire and its middleware are being run in Windows Subsystem for Linux (WSL). This way, all of the components can run in the same machine. The middleware communicates with ForeFire using ForeFire's shell commands.

The middleware communicates with the Disturbances Manager using a secure RabbitMQ connection, which the work of Costa possibilitated [Costa, 2020]. They communicate using a single exchange in which two different queues exist: one is named *ForeFireRecv*, in which the ForeFire middleware receives commands, and the other is named *ForeFireSend*, in which ForeFire replies to the commands it receives. The commands are sent in plaintext, with the following format: "<*COMMAND*> <*arg0*> <*arg1*> <...>". The commands that the middleware can receive are the following:

- *INIT* <*left*> <*right*> <*top*> <*bottom*> – Initializes the landscape area for a given bounding box for the fires' locations. With this, the middleware can calculate the optimal area to be simulated in ForeFire and then requests the weather information to interpolate the wind data. It does this by calculating the necessary QMID tiles to enclose this bounding box, and then surrounding this grid with a layer of QMID tiles. The number of layers used to surround the area is definable.

- *WIND* <*airport*> <*heading*> <*speed*> <*unit*> – Receive the wind information for a given weather station.

- *START* <*timestamp*> – Writes the landscape data and starts the ForeFire simulation, as soon as possible, with the simulated time in the received timestamp.

- *FIRE* <*latitude*> <*longitude*> <*time since start*> – Indicates the location of a fire and when to start it.

- *STEP* – Tells the middleware to retrieve another ForeFire simulation step.

At the time the middleware receives the *START* command, it uses the *setParameters* command in the ForeFire shell to initialise required simulation parameters, like the data paths, simulation resolution, and output mode. After that, when it receives one or more *FIRE* commands, it sends them to the ForeFire shell using the *startFire* command. For the *STEP* command, the middleware sends the *step* command to the ForeFire shell with the wanted time delta, which is one second in this case. Because ForeFire works based on events, the time progress might a scheduled event (which will, in turn, schedule further ones). After the command is sent, one of two things can

[5] Pexpect is available at https://pexpect.readthedocs.io/en/stable/index.html.

happen: either an event occurred and ForeFire prints information related to it to the pseudoterminal or no new event occurred, in which case ForeFire outputs nothing. If ForeFire responds with a fire event, the middleware then sends the command *print*, to which the ForeFire shell replies with a JSON object containing a timestamp, the burnt area and the area polygon defined by its vertexes. However, if ForeFire didn't respond after the timestep, the *step* command is simply sent again until an event occurs and the simulation progresses.

As for commands that the middleware can send, they are the following:

- *WIND <airport0> <airport1> <...>* – Requests the wind information for a certain airport list, if there's an associated station with that airport. It then receives each of them individually, and after a short period without receiving any information (100 milliseconds), it assumes there's no more information to be received.

- *READY* – After interpolating the wind data and preparing all the landscape, it sends this message to signal it is prepared to start the simulation.

- *STEP FRONT <timestamp0> <area0> <vertex00> <vertex01> <vertex0...> FRONT <timestamp1> <area1> <vertex10> <vertex11> <vertex1...> <...>* – Returns a simulation step, indicating the fire fronts' timestamps, the affected areas, and the polygons that defines those areas, via their vertexes.

At some time, the Disturbances Manager will make a request for FSX, using SimConnect, to send it the simulation's time periodically (currently set for one second), which it will then use to compare to the steps' timestamps.

An important aspect to mention is that while FSX utilises WGS 84 (used in global positioning system) as its projection, which uses geodesic coordinates, ForeFire needs to have a conformal projection with coordinates in metres [NIMA, 2004]. Because of this, the World Mercator projection, identified by the code EPSG:3395, was chosen [EPSG, 2020b]. It has certain limitations, such as being limited to 85 degrees North and South. This is irrelevant, however, when we consider that fire would not propagate on water or ice, which is what the entire non-included region consists of. This conversion is used for the preparation of the domain data to go into the landscape file, and later on in the simulation to convert the vertexes of the polygon to return to the Disturbances Manager. However, converting coordinates between projection is not trivial, and as such, the PROJ[6] software is used via its Python module, pyproj. The conversion of many vertexes, the number of which increases as the affected area is bigger, is more and more time expensive. The impact of this increasing delay is further explored later on. Converting in the opposite direction, i.e. converting the agents' telemetry to the projection used by ForeFire is also not feasible since the number of agents can be bigger than the number of vertexes that define the affected area. The agents' telemetry would also have to be converted at a faster frequency than the simulation steps, which often have a duration of several seconds. Finally, tools to display information on maps

[6]PROJ is available at `https://proj.org/`.

normally use WGS 84 compatible projections (which the case for Google Maps, used on The Platform), like the Web Mercator projection [EPSG, 2020a]. However, they aren't compatible with the World Mercator projection.

When the Disturbances Manager receives the simulation steps from ForeFire, it then uses the basis of the work of Almeida to add the fire area to a disturbance's component and check if the simulated agents are affected by that disturbance's component [Almeida, 2017]. The Disturbances Manager buffers the simulation steps in a queue, always having a certain number of steps in advance, to prevent hitching. The optimal number of buffered simulation steps is studied later on, taking into account the simulation speed that FSX can support. The Disturbances Manager is getting periodic timestamps of the simulation time. Whenever the simulation time is equal or greater than the simulation step in front of the queue's timestamp, that step is dequeued, its information applied to the disturbance's component and a new simulation step is requested to restore the buffer's optimal size.

Figure 5.6 shows the typical communication timeline for all the components of the co-simulation architecture.

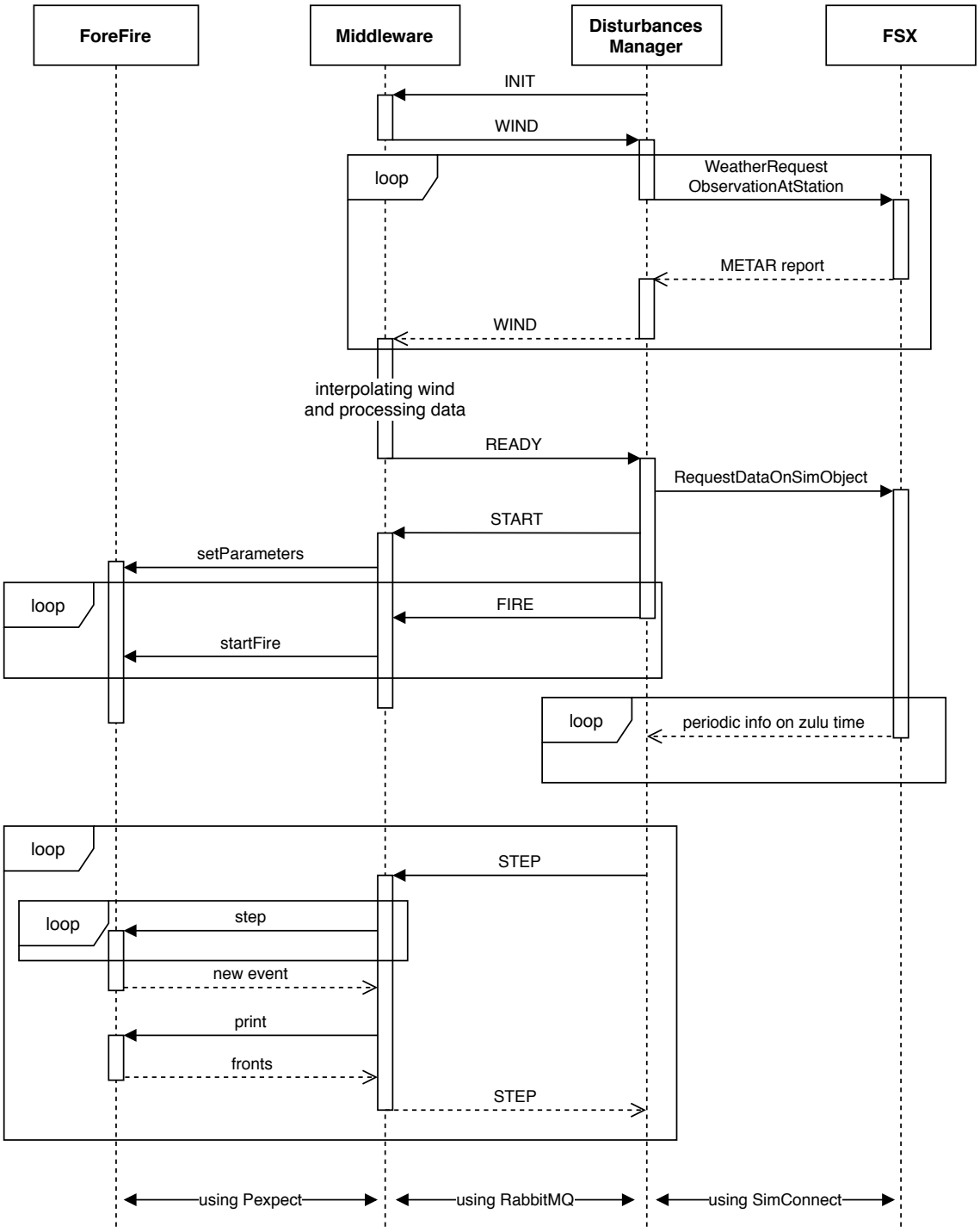


Figure 5.6: Typical Co-Simulation Communication Timeline

Chapter 6

Testing and Results

In this section, the testing of the implementation and the obtained experimental results are displayed and analysed. Different implementation aspects and metrics are taken into consideration. Firstly, a functional test is shown and afterwards, the solution's performance is analysed.

6.1 Functional Test

To test that the implementation was working, a scenario was devised where five agents, flying Cessnas 172SP Skyhawk (which can be seen in Fig. 6.1) would come out of John Glenn Columbus International Airport (ICAO code KCMH), in Ohio, with a heading of 180 degrees (heading south), at their cruise speed of 120 knots. This agent team was defined used the before mentioned Team Description Language, which specifies the static information about the team [Silva et al., 2017]. The agents' payloads contained two sensors each, a *Chemical* sensor (to measure CO_2) and a *CameraIntensity* one. These aeroplanes would keep their heading for 70 kilometres, at which time they would get to the Ross Lake Wilderness Area. In this area, a circle with a 3000 feet radius was defined, and a fire would start in a random location within the circle.

In the established scenario, we could see the simulated fire spread inside of the Disturbances Manager built-in map, as the agents approached the fire spawn area. Figure 6.2a shows the Disturbances Manager GUI, where it can be seen that the aeroplanes are approaching a burning area.

No aircraft have been affected by the fire disturbance, and no messages have been sent from the Disturbances Manager to any agents. Other data available in the GUI are the aeroplanes telemetry and the current disturbances' components.

After a while, one of the agents gets to the area affected by the fire disturbance. At this time, the Disturbances Manager marks it as affected by the component, because the agent contains a sensor which is compatible with the reading simulated by the fire component (in this instance, *CameraIntensity*). This can be seen in Fig. 6.2b.

Figure 6.1: Simulated Cessna 172S Skyhawk in FSX

Afterwards, the agent exits the affected area, and is no longer sent any messages or marked as affected by the disturbance. Figure 6.3 shows this situation. The figure also shows that the Disturbances Manager sent various messages with readings of the aforementioned sensor to that agent. The *s10* sensor is correctly corresponding to the *CameraIntensity* sensor of that agent.

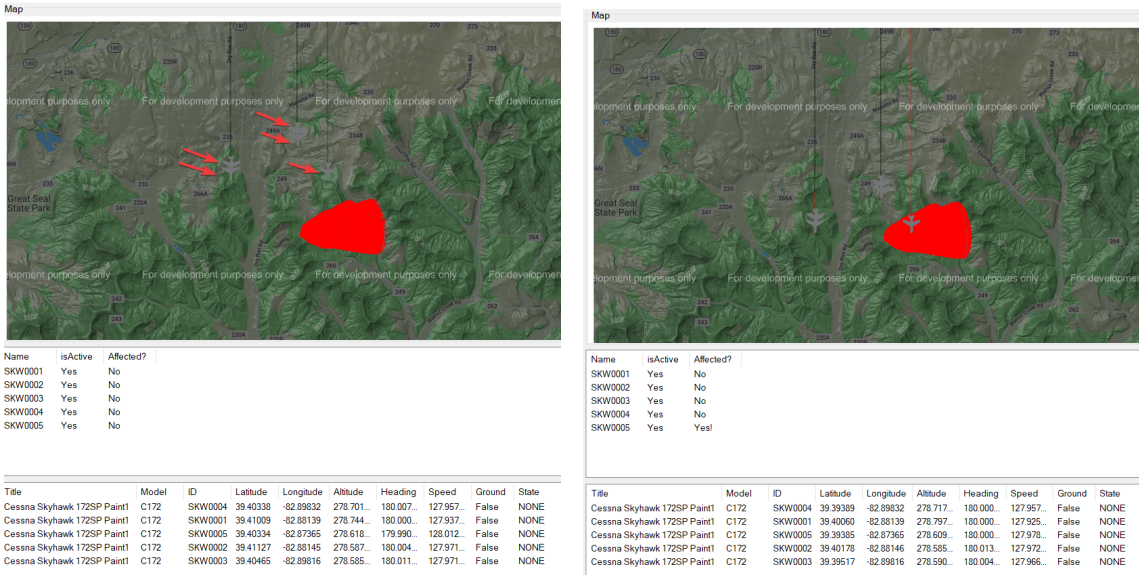
Knowing that the solution works, it is then needed to study, objectively, how well it works.

6.2 Performance Analysis

In this section, the performance of the implemented solution is studied, taking into account aspects like the simulation speed, time drift, data compatibility and latency. To test the performance of the solution, a little over an hour of fire simulation data was gathered and analysed. That experimental data is used in this section and is available, in its entirety, in Appendix B. Experiments I and II were performed with the exact same conditions, to rule out possible variations in processing time. They had the same randomly selected start location, and a wind of 15 knots at 135 degrees. Experiment III doubles the wind speed to 30 knots. Experiment IV has the same conditions as experiments I and II, but a larger ForeFire domain. Experiment V uses the conditions in experiments I and II for two simultaneous fire fronts.

6.2.1 Data Compatibility

As mentioned before in section 5.2, the map projections used by ForeFire are not the same as those used by FSX. PROJ was used to convert the coordinates from one projection to the other. For an average of 100 runs, the time it takes to convert between the World Mercator projection and WGS



(a) Approach Phase (b) Affected Phase

Figure 6.2: Disturbance Manager's View. The red arrows point to the aeroplanes, while the red area defines where the fire has spread to.

84 is 32.835 milliseconds. However, this time scales linearly with the number of coordinate pairs to convert. For the same 100 runs, Fig. 6.4 shows the time to convert up to 500 coordinate pairs. On average, a single coordinate pair took 328 milliseconds to be converted.

To try and counter part of the impact that the conversion will surely have, all of the projection conversion is performed on a different thread from the main middleware thread, in order to allow for the simulation to continue while the polygons are being converted.

The data given to the fire simulator for the elevation is, one to one, the same present in FSX. The fuel data and wind had to be slightly changed, as described in sections 5.1.4 and 5.1.6.

For the fuel data, some detail that is present on the Olson Land Cover system is lost when converting to CLC. The former has more micro-oriented information for the land, especially in specific vegetation types. However, because CLC is more oriented for the land purpose, some present land types cannot be found in FSX's data. This means that, unfortunately, there's a loss of information derived by the fact that the land systems were created with different purposes.

As for the wind, the implemented interpolation system is described in the aforementioned section. It is working well and, for the mostly uniform wind patterns that FSX generates, it works perfectly.

6.2.2 Simulation Speed

First, we'll consider experiments I and II. Figure 6.5 shows how the simulation speed, otherwise defined as the ratio between the simulated time and the real elapsed time, progressed along the time steps, in Experiment I. Experiment II has shown no significant variance from the results obtained in experiment I. It can be observed that it fluctuates a lot, but usually tops around 6x the real-time,

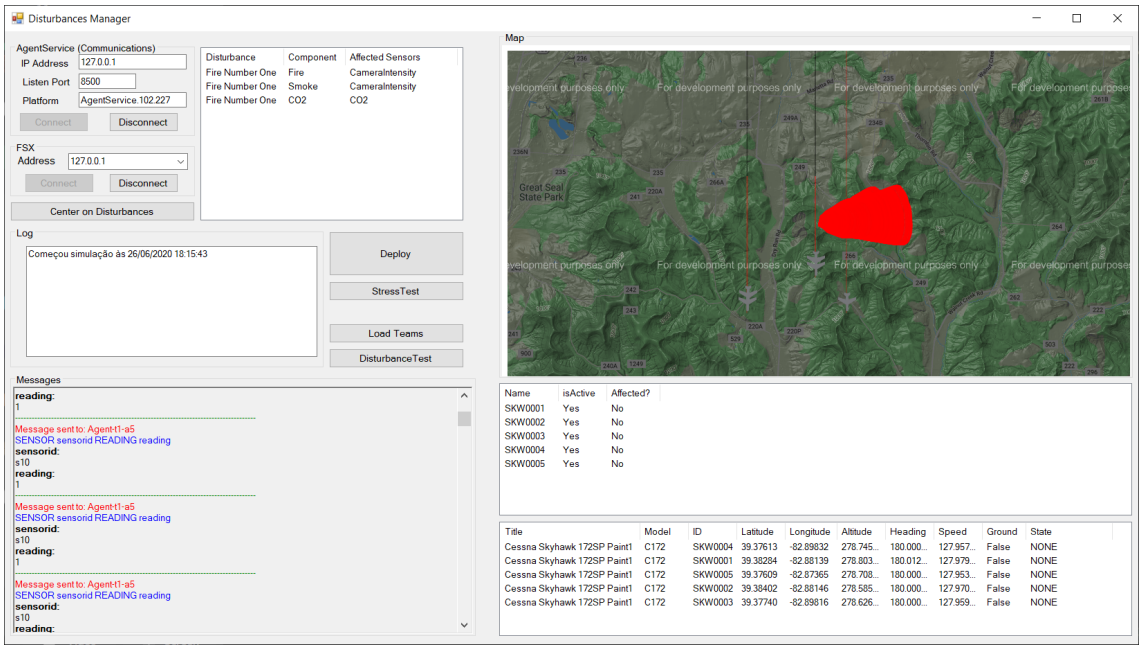


Figure 6.3: Disturbance Manager's View of Agents After Being Affected

and mostly above 2x the real-time. However, this is an issue, seeing that FSX can simulate at up to 4x the real speed (though only in steps of 1x, 2x or 4x), and it would be good to attain the same simulation speed in the implemented solution.

We can determine the impact of the projection conversion by measuring how long the projection conversion takes. First of all, the number of vertexes increases as the simulation time passes (Fig. 6.6). From the previous section, it was learned that it will have an increasing impact on the simulation steps. By subtracting the time it took to convert the projections in each step, named *conversion time*, from the true time it took to calculate the step, named *real time*, we can get the actual duration spent solely on the simulation. With this value, we can calculate the simulation speed if the projection conversion wasn't required. This is shown in Fig. 6.7, where the simulation speed now rarely drops below 4x.

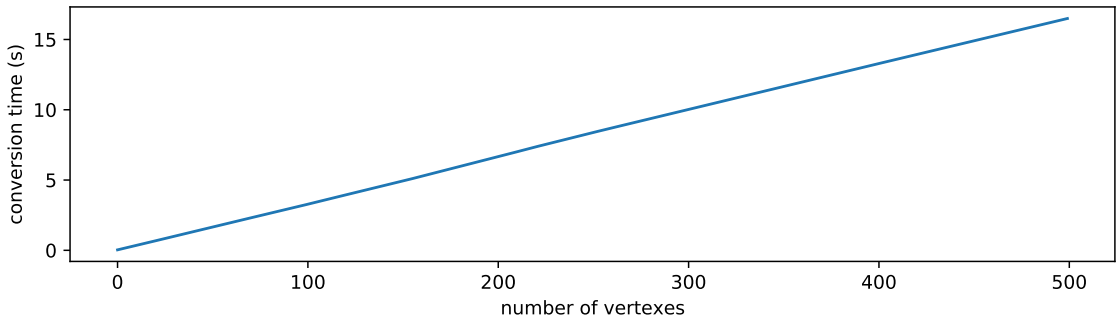


Figure 6.4: Time to Convert Coordinates Between Projections

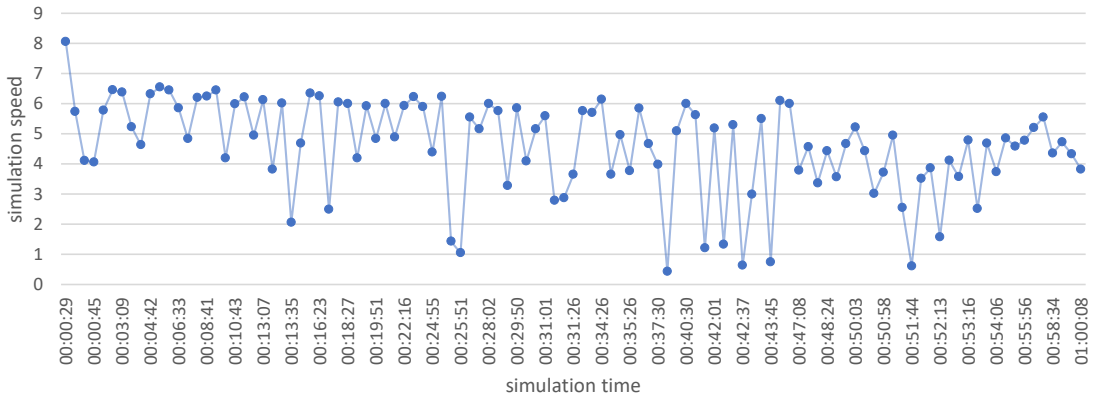


Figure 6.5: Simulation Speed in Experiment I

By calculating the percentage of the time that is used for converting projections for each step, and then sorting them by descending percentage, it can be seen that, bar outliers, small simulated time steps are the ones that are hit the most by the conversion time, with around 90% of the time those steps took to be calculated being spent solely converting the projection. The top and bottom 5 percentages are shown in Table 6.1. Another factor to take into account is that the printing of simulation data also incurs a small overhead, which further affects the smaller time steps.

Table 6.1: Projection Conversion Percentage in Experiment I

Simulation Time	Step Time (s)	Real Time (s)	Conversion Time (s)	Percentage
00:51:44	1	1.639572	1.534392	93.58%
00:42:37	1	1.570443	1.46608	93.35%
00:25:51	1	0.953864	0.847916	88.89%
00:37:31	1	2.338254	1.950626	83.42%
00:42:03	2	1.508573	1.251898	82.99%
...	...	...	...	...
00:11:57	74	11.90522	0.441005	3.70%
00:00:29	29	3.600048	0.117649	3.27%
00:08:41	72	11.5264	0.37353	3.24%
00:03:09	54	8.46984	0.25928	3.06%
00:02:15	72	11.15213	0.226867	2.03%

The time synchronisation of the simulator and the Disturbances Manager is done by requesting time a certain number of steps in advance. The size of this buffer is essential to guarantee that the simulation doesn't dip under a certain simulation speed.

In Table 6.2, the simulation speeds for the buffer of sizes 1 to 10 are shown for the first 10 steps. They are calculated by dividing, for a buffer of size n, the sum of the current and previous $n-1$ step simulation times by the sum of the current and previous $n-1$ steps real times, i.e. the total simulated time divided by the real time it took to simulate it in the current and last $n-1$ steps. This calculates the effective buffered simulation speed, since it won't matter if a single step

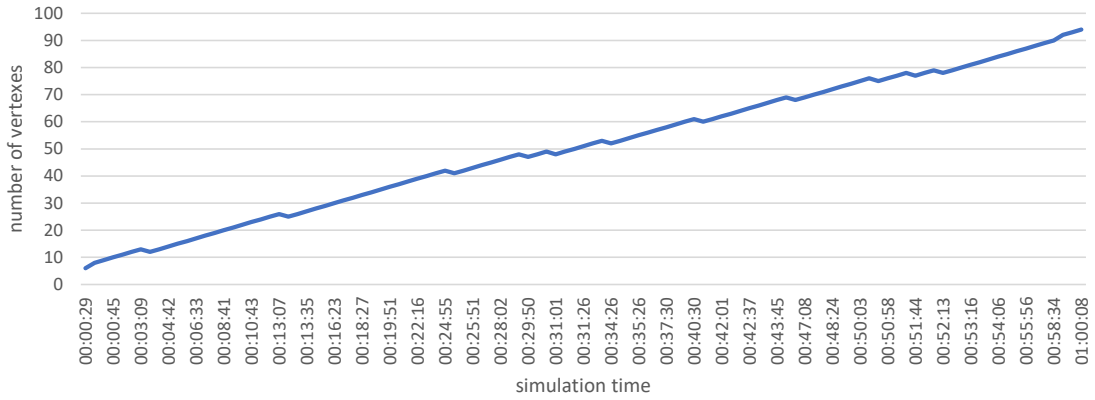


Figure 6.6: Number of Vertexes in Experiments I and II

has a really high or low simulation speed, but how every step affects the simulation speed in the buffer. The missing values mean that the buffer wasn't yet at its full capacity. In those cases, the simulation speed would be equal to the one in the buffer of size $n-1$. It is immediately noticeable that, even though certain steps had a low simulation speed, the fact they occur in smaller time steps means they are easily absorbed by the higher simulation speeds on bigger steps.

Calculating the average simulation speed for all the time steps in the experiment, we can see what the improvement from the previous buffer size is. In Table 6.3, this improvement is calculated and, from buffer size 11 onwards, the improvement is pretty much the same. For this reason, a buffer size of 10 is ideal, since increasing the buffer size above this number would yield diminishing returns.

The simulation speed with the selection of a 10 step buffer can be seen in Fig. 6.8. The result is very improved upon the one in Fig. 6.5, where no buffer is taken into account, and closer to the one on Fig. 6.7, where no projection conversion happens. However, the fact that the conversion needs to happen and does happen in Fig 6.8 can be noticed with the slight downward trend at the end. In any case, the fact that it has an average speed above 4x, even though it dips under this speed

Table 6.2: Simulation Speeds by Buffer Size

s1	s2	s3	s4	s5	s6	s7	s8	s9	s10
8.0554	—	—	—	—	—	—	—	—	—
5.7347	7.4678	—	—	—	—	—	—	—	—
4.1111	4.9244	6.7916	—	—	—	—	—	—	—
4.0595	4.0880	4.6754	6.4082	—	—	—	—	—	—
5.7800	5.3664	5.0792	5.2016	6.2152	—	—	—	—	—
6.4562	6.3086	6.1633	6.0117	5.9926	6.3414	—	—	—	—
6.3756	6.4214	6.3335	6.2391	6.1353	6.1166	6.3511	—	—	—
5.2247	6.0990	6.2776	6.2166	6.1361	6.0469	6.0337	6.2581	—	—
4.6306	5.0799	5.9934	6.2162	6.1645	6.0884	6.0040	5.9929	6.2159	—
6.3170	6.2026	6.0327	6.1542	6.2503	6.2119	6.1578	6.0961	6.0855	6.2425

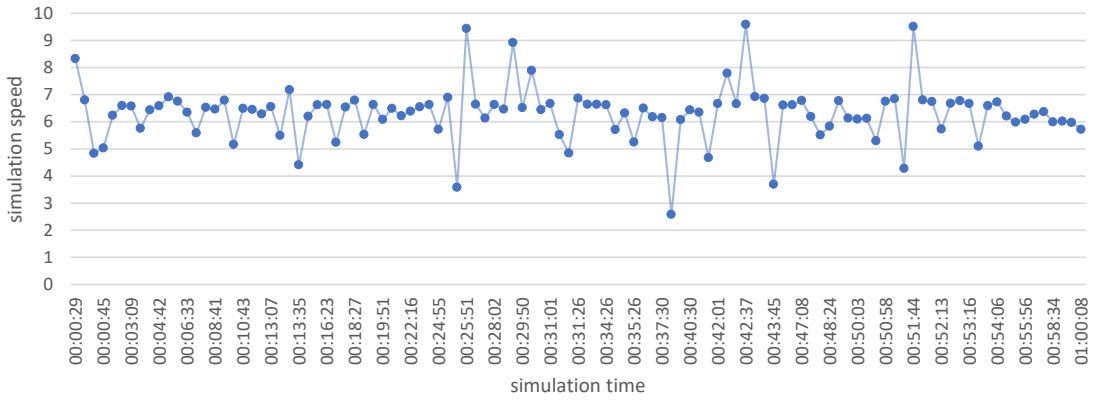


Figure 6.7: Simulation Speed Without Projection Conversion in Experiment I

for a bit around the 52 minute mark, and can keep constantly above 2x speed is very positive.

Experiment III shows us that increasing the wind leads to a faster propagation of the fire. In these cases, more time steps are generated for the same simulation time. When compared to experiment I, 1.734 times the amount of steps were generated in the same timeframe. This has, unfortunately, a significant effect on the performance of the implemented solution. While in experiment I a total of 5529 coordinates were converted, the number increased to 17485 in experiment III. Applying the same analysis that was done on the previous experiment, the ideal buffer size is smaller, at only 7 steps. This happens because the performance degrades at a much faster rate and increasing the buffer size does not further allow for previous and next steps to compensate for a slow step. The simulation speed for this experiment can be seen in Fig. 6.9. Generalising the results of this experiment, it can be concluded that changes in simulation conditions that increase the fire spread rate will have a great impact on the implemented solution's performance.

Experiment IV was made to analyse the impact of the defined fire dimension on the simulation.

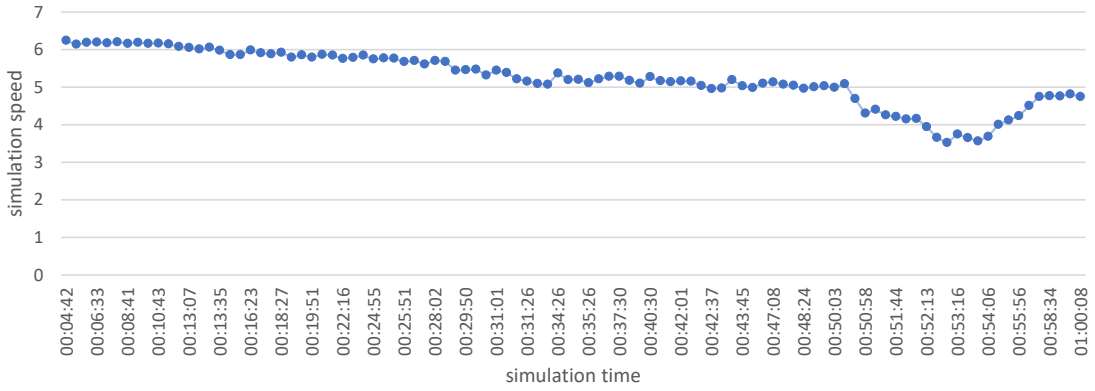


Figure 6.8: Simulation Speed With 10 Step Buffer in Experiment I

Table 6.3: Improvement in Average Simulation Speed by Buffer Size

Buffer Size	Average Simulation Speed	Improvement
1	4.616897	—
2	4.989636	8.07%
3	5.099315	2.20%
4	5.154257	1.08%
5	5.192315	0.74%
6	5.207528	0.29%
7	5.220316	0.25%
8	5.228226	0.15%
9	5.234754	0.12%
10	5.239840	0.10%
11	5.241240	0.03%
12	5.239909	-0.03%
13	5.240043	0.00%
14	5.241431	0.03%

While experiments I and II had their bounding boxes surrounded by 1 QMID layer (i.e. the QMID tile that contained the bounding box was surrounded by 1 level of tiles, all around), experiment IV was instead surrounded by 2 QMID layers. It shows that changing the defined dimension does not alter the performance of the simulation. However, it does change how long the interpolation takes. Because of this, this parameter needs to be manually fine-tuned for each scenario, taking into account the fire conditions (fast or slow spread), the length of the experiment and the required interpolation time. Some differences in the steps appeared after a while when compared to experiments I and II. These differences can be attributed to the slightly different alignment of the wind map after the interpolation.

Experiment V had two fire fronts in conditions similar to experiments I and II. The fronts can be seen in Fig. 6.10. An immediate observation can be made that because twice the amount of vertexes in each step needed to be converted, smaller time steps took a huge hit in their simulation

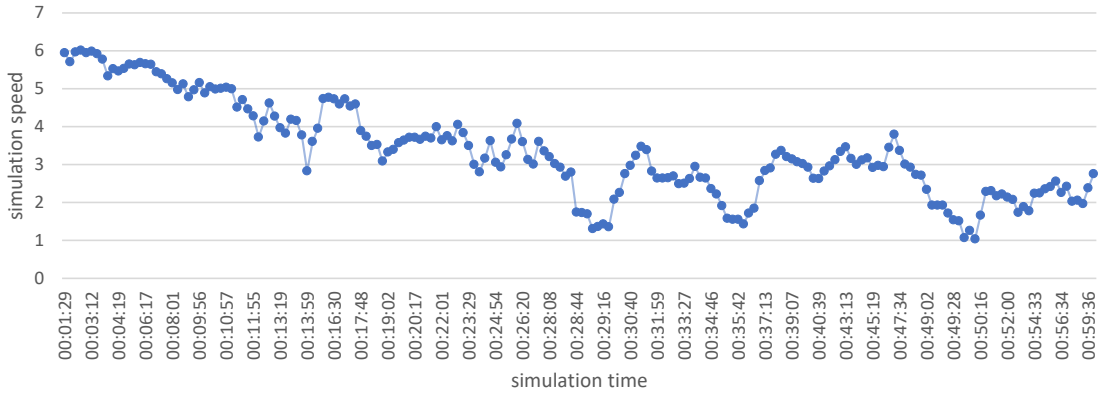


Figure 6.9: Simulation Speed With 7 Step Buffer in Experiment III

Figure 6.10: Disturbance Manager's View of Two Fire Fronts

speed, frequently dipping under 1x. This can be seen in Fig. 6.11. After analysis, the ideal buffer size was determined to be roughly the same as experiments I and II (with improvements stagnant after a 10 step buffer). With this buffer size applied, the simulation speeds are, however, similar to the ones in experiment III, and can be seen in Fig. 6.12. This might suggest that on one hand, the buffer size depends on how fast the spread is and not directly the number of vertexes. On the other hand, the simulation speed is indeed limited by the increasing number of vertexes.

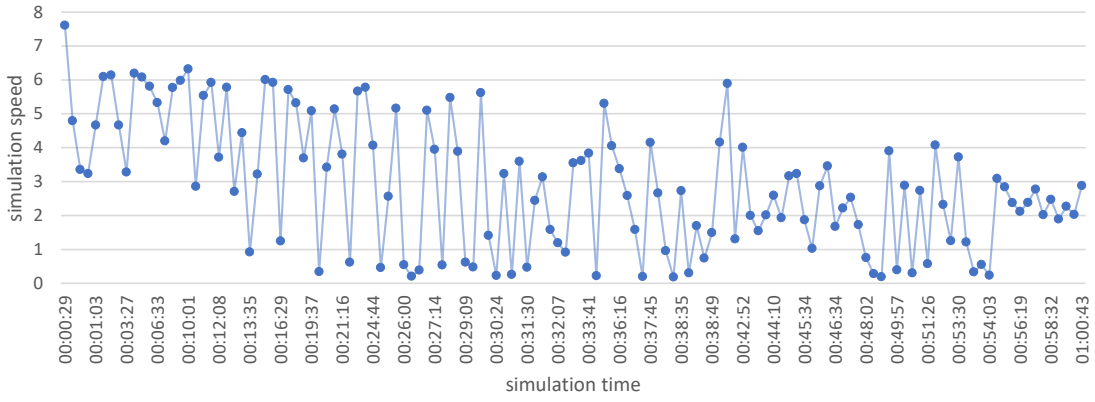


Figure 6.11: Simulation Speed in Experiment V

6.2.3 Information Latency

The implemented solution of keeping the steps in the Disturbances Manager ahead of time means that, technically, there is no latency involved in regards with the Disturbances Manager applying the simulation step to the simulation environment. In practice, however, there are always small overheads in the way that the Disturbances Manager currently processes the incoming FSX agent telemetry and then applies the disturbance data to the affected vehicles. However, these overheads

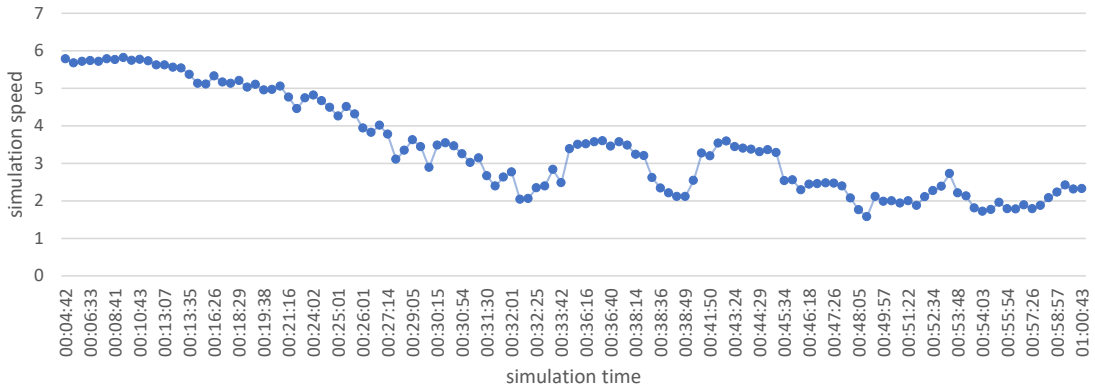


Figure 6.12: Simulation Speed With 10 Step Buffer in Experiment V

are neither significant, in the aspect that they amount to some computational cycles, nor caused by the present solution of the co-simulation architecture.

6.2.4 Time Drift

When it comes to time drift, the Disturbances Manager consumes the queue whenever the timestamp of the front step is equal or lower than the current time. This means that, if the simulation is running late, the Disturbances Manager always tries to catch-up.

However, the amount of existing drift will be highly dependent on the simulation speed that The Platform is running at. At 1x and 2x the real-time, using the implemented buffer solution, there is no drift involved in slow spread simulations. In any case, for the highest simulation speed of 4x the real-time, even though the solution speed is, on average, above 4x, sometimes the buffered simulation speed will dip under that value. If there is a fast fire spread, only the simulation speed of 1x is achieved throughout the experiment, albeit having an average speed above 2x.

Because of the catch-up mechanic described before, this means that even though time drift can occur at the highest simulation speed, it will eventually correct itself as the next simulation steps come in faster. The exact amount of time drift will always depend on the simulation scenario, increasing the longer the simulation goes on because of the current projection conversion system. For the simulation speed described in Fig. 6.8, there's an average of a -124.0% simulation speed drift respective to a 4x simulation, meaning that effectively, it is running faster than required. However, the average for the whole experiment does not paint the whole story, since, in the beginning, the simulation is much faster than in later stages. For example, when considering only the values from the latter half of the experiment (from 00:52:13 to 00:54:06, where the dip occurred), an average simulation speed drift of 31.8% can be seen, when comparing to the 4x simulation speed.

Chapter 7

Conclusions and Future Work

This section will list the conclusions that can be taken from the implemented solution and its analysis, and future work that can improve upon that solution.

7.1 Conclusions

The increasing number of autonomous vehicles and environmental disturbances means that their interaction needs to be better studied, in order to make these autonomous vehicles interact with different kinds of disturbances. Previous works implemented the Disturbances Manager, where disturbances were defined and readings for their components were generated [Almeida, 2017]. To get a more realistic simulation and to implement disturbances that could not be reproduced inside the Disturbances Manager and FSX, a need to create a co-simulation architecture to add an external tool to simulate such disturbances existed.

The literature review found that, in the need of implementing such an architecture, most related works used their own ad-hoc implementations, despite standards such as HLA or FMI existing. These standards are somewhat complex and perhaps better fitted for creating a simulation environment from the ground up, as adapting the entire Platform's architecture would be a massive undertaking. Some alternatives, like DDS, appeared to be a good solution, but available implementations of such standard were found to be lacklustre. To standardise the communication architecture used in various components of the platform, RabbitMQ was chosen to perform that task.

After a schematisation of different kinds of disturbances, and then a study of different simulators, a fire simulator, ForeFire, was chosen to simulate a fire disturbance within The Platform. Input retrieval from FSX and processing was required to extract the data from the scenery files for various components, like fuel, which was based on land classification, elevation and wind data,

which required interpolation. Coordinate systems also needed to be converted to be used in ForeFire. Using a Python middleware, the communication between the simulator and the Disturbances Manager was facilitated.

The implemented solution was tested and found to work. Several aspects of the performance of this solution were analysed, and optimal parameters for the solution were found. The solution runs in slow fire spread with 1x and 2x the real-time simulation speed. In the initial stages of the fire spread the simulation speed is steadily above 4x. In later stages of the simulation, a low value of time drift occurs, as the simulation speed dips under 4x. For fast fire spread scenarios, the 1x speed is reached, and 2x speed can be used with some time drift.

7.2 Future Work

To further improve upon the developed work, the main aspect to take into consideration is the development of better methods to create compatibility between the map projections between FSX and ForeFire. The conversion between projections represents a major impact in the time a simulation step requires to be calculated but is not a fault of fire simulator speed itself. Reworking the simulator to be able to use non-conformal projections with geodesic coordinates is a possibility, whereas creating a better conversion system, which has faster performance and further parallelises operations, is another. A different take on this issue can be to use larger time steps (e.g. of one minute and above), whose fast simulation speeds will absorb the conversion time, at the cost of simulation realism at a micro time scale. Then, interpolation between step areas could be performed directly on FSX's projection to restore smaller time steps, therefore reducing the overall number of projection conversions.

Time synchronisation can be improved by having the Disturbances Manager detect when a simulated disturbance's component is running behind in simulation time and holding the FSX simulation nodes until the component's simulation is up to speed. The implemented buffer mechanism can also be improved by automating the analysis of the simulation speed and adjusting its parameters automatically. Other forms of buffering can also be analysed, i.e. using a buffer of a certain time period instead of a number of simulation steps.

Improving the compatibility in the land classification systems is also a possibility. One can achieve this by reworking the existing data from FSX to add relevant land classes from CLC, or by using CLC, which classified the entire European Union using that system, within FSX.

Changing the wind interpolation methods to get even more realistic results can also be achieved by using methods that better take into account the wind behaviour, albeit taking into account the performance impact that such an interpolation scheme would have.

The current simulated sensors for detecting the fire only get triggered in a binary form and when inside the disturbance area, despite it being an intensity sensor. Improving the way that the sensor detects the fire, by having a range in which the sensor can detect the fire component, or using other models of the simulator to calculate emitted heat or particles, in order to get an intensity value for the fire, would further approximate the simulation to its definition.

Finally, this work proves that an external simulator for a certain disturbance can be successfully integrated into The Platform. This advance allows for the possibility of including further external simulator for other types of disturbances, bringing even more realism to the disturbances to be simulated within The Platform's missions.

References

[Albagli et al., 2016] Albagli, A. N., Falcão, D. M., and de Rezende, J. F. (2016). Smart grid framework co-simulation using HLA architecture. *Electric Power Systems Research*, 130:22–33. DOI: 10.1016/j.epsr.2015.08.019.

[Albini, 1979] Albini, F. A. (1979). Spot fire distance from burning trees: a predictive model. General Technical Report INT-56, US Department of Agriculture, Forest Service, Intermountain Forest and Range Experiment Station.

[Alexander, 1993] Alexander, D. (1993). *Natural Disasters*. Routledge. ISBN: 978-1-85728-093-7.

[Allan, 2000] Allan, J. R. (2000). The costs of bird strikes and bird strike prevention. In Clark, L., Hone, J., Shivik, J. A., Watkins, R. A., VerCauteren, K. C., and Yode, J. K., editors, *Proceedings of the 3rd National Wildlife Research Center (NWRC) Special Symposium, August 1–3 2000, Fort Collins, Colorado, USA*, page 18.

[Almeida, 2017] Almeida, J. F. d. S. (2017). Simulation and Management of Environmental Disturbances in Flight Simulator X. Master's thesis, Faculty of Engineering, University of Porto, Porto, Portugal.

[Awais et al., 2013] Awais, M. U., Mueller, W., Elsheikh, A., Palensky, P., and Widl, E. (2013). Using the HLA for Distributed Continuous Simulations. In *Proceedings of the 2013 8th EUROSIM Congress on Modelling and Simulation, September 10–12 2013, Cardiff, United Kingdom*, pages 544–549. DOI: 10.1109/EUROSIM.2013.96.

[Balbi et al., 2009] Balbi, J., Morandini, F., Silvani, X., Filippi, J., and Rinieri, F. (2009). A physical model for wildland fires. *Combustion and Flame*, 156(12):2217–2230. DOI: 10.1016/j.combustflame.2009.07.010.

[Bauer and Van Duijsen, 2005] Bauer, P. and Van Duijsen, P. J. (2005). Challenges and advances in simulation. In *Proceedings of the 2005 IEEE 36th Power Electronics Specialists Conference (PESC 2005), June 12–18 2005, Recife, Brazil*, pages 1030–1036. DOI: 10.1109/PESC.2005.1581755.

[Bian et al., 2015] Bian, D., Kuzlu, M., Pipattanasomporn, M., Rahman, S., and Wu, Y. (2015). Real-time co-simulation platform using OPAL-RT and OPNET for analyzing smart grid performance. In *Proceedings of the 2015 IEEE Power Energy Society General Meeting, July 26–30 2015, Denver, Colorado, USA*, pages 1–5. DOI: 10.1109/PESGM.2015.7286238.

[Blochwitz et al., 2011] Blochwitz, T., Otter, M., Arnold, M., Bausch, C., Elmqvist, H., Junghanns, A., Mauß, J., Monteiro, M., Neidhold, T., Neumerkel, D., Olsson, H., Peetz, J.-V.,

Wolf, S., and Clauß, C. (2011). The Functional Mockup Interface for Tool independent Exchange of Simulation Models. In *Proceedings of the 8th International Modelica Conference, March 20–22 2011, Dresden, Germany*, number 63 in Linköping Electronic Conference Proceedings, pages 105–114. Linköping University Electronic Press; Linköpings universitet. DOI: 10.3384/ecp11063105.

[Bossard et al., 2000] Bossard, M., Feranec, J., and Otahel, J. (2000). CORINE land cover technical guide – Addendum 2000. Technical Report No 40/2000, European Environment Agency, Copenhagen.

[Bragard et al., 2017] Bragard, Q., Ventresque, A., and Murphy, L. (2017). Self-balancing decentralized distributed platform for urban traffic simulation. *IEEE Transactions on Intelligent Transportation Systems*, 18(5):1190–1197. DOI: 10.1109/TITS.2016.2603171.

[Brito et al., 2015] Brito, A. V., Bucher, H., Oliveira, H., Costa, L. F. S., Sander, O., Melcher, E. U. K., and Becker, J. (2015). A Distributed Simulation Platform Using HLA for Complex Embedded Systems Design. In *Proceedings of the 2015 IEEE/ACM 19th International Symposium on Distributed Simulation and Real Time Applications (DS-RT), October 14–16 2015, Chengdu, China*, pages 195–202. DOI: 10.1109/DS-RT.2015.16.

[CALCHAS, 2010] CALCHAS (2010). Development of an Integrated Analysis System for the effective fire conservancy of forests. Program LIFE08ENV/GR/000558, LIFE + (Environment Policy and Governance).

[Cellura et al., 2017] Cellura, M., Guarino, F., Longo, S., and Mistretta, M. (2017). Modeling the energy and environmental life cycle of buildings: A co-simulation approach. *Renewable and Sustainable Energy Reviews*, 80:733–742. DOI: 10.1016/j.rser.2017.05.273.

[Chombart, 2012] Chombart, P. (2012). Multidisciplinary modelling and simulation speeds development of automotive systems and software. Innovation report, Information Technology for European Advancement.

[Cicirelli et al., 2009] Cicirelli, F., Furfaro, A., and Nigro, L. (2009). An Agent Infrastructure over HLA for Distributed Simulation of Reconfigurable Systems and Its Application to UAV Coordination*. *SIMULATION*, 85(1):17–32. DOI: 10.1177/0037549708100187.

[Cicirelli et al., 2015] Cicirelli, F., Giordano, A., and Nigro, L. (2015). Efficient environment management for distributed simulation of large-scale situated multi-agent systems. *Concurrency and Computation: Practice and Experience*, 27(3):610–632. DOI: 10.1002/cpe.3254.

[Costa, 2020] Costa, D. H. F. (2020). Secure Communication in a Distributed Simulation Platform. Master's thesis, Faculty of Engineering, University of Porto, Porto, Portugal.

[CRED, 2015] CRED (2015). The human cost of natural disasters - a global perspective. Technical report, Centre for Research on the Epidemiology of Disasters (CRED).

[Dahmann et al., 1998a] Dahmann, J. S., Fujimoto, R. M., and Weatherly, R. M. (1998a). The DoD High Level Architecture: an update. In *Proceedings of the 1998 Winter Simulation Conference, December 13–16 1998, Washington, DC, USA*, volume 1, pages 797–804. DOI: 10.1109/WSC.1998.745066.

[Dahmann et al., 1998b] Dahmann, J. S., Kuhl, F., and Weatherly, R. M. (1998b). Standards for Simulation: As Simple As Possible But Not Simpler The High Level Architecture For Simulation. *SIMULATION*, 71(6):378–387. DOI: 10.1177/003754979807100603.

[Dahmann and Morse, 1998] Dahmann, J. S. and Morse, K. L. (1998). High Level Architecture for simulation: an update. In *Proceedings of the 2nd International Workshop on Distributed Interactive Simulation and Real-Time Applications, July 20 1998, Montreal, Quebec, Canada*, pages 32–40. DOI: 10.1109/DISRTA.1998.694563.

[D'Angelo et al., 2016] D'Angelo, G., Ferretti, S., and Ghini, V. (2016). Simulation of the Internet of Things. In *Proceedings of the 2016 International Conference on High Performance Computing Simulation (HPCS), July 18–22 2016, Innsbruck, Austria*, pages 1–8. DOI: 10.1109/HPCSim.2016.7568309.

[Dols et al., 2016] Dols, W. S., Emmerich, S. J., and Polidoro, B. J. (2016). Coupling the multizone airflow and contaminant transport software CONTAM with EnergyPlus using co-simulation. *Building Simulation*, 9(4):469–479. DOI: 10.1007/s12273-016-0279-2.

[EPSG, 2020a] EPSG (2020a). WGS 84 / Pseudo-Mercator. GeoRepository. Available online at https://epsg.org/crs_3857/WGS-84-Pseudo-Mercator.html. Last seen in 2020-06-09.

[EPSG, 2020b] EPSG (2020b). WGS 84 / World Mercator. GeoRepository. Available online at https://epsg.org/crs_3395/WGS-84-World-Mercator.html. Last seen in 2020-06-09.

[Esteves, 2020] Esteves, J. F. V. D. (2020). Distributed Simulation and Exploration of a Game Environment. Master's thesis, Faculty of Engineering, University of Porto, Porto, Portugal.

[Evans, 2014] Evans, J. K. (2014). An updated examination of aviation accidents associated with turbulence, wind shear and thunderstorm. Technical Report AMA Report Number 14-14, Analytical Mechanics Associates, Inc.

[Filippi et al., 2014] Filippi, J.-B., Bosseur, F., and Grandi, D. (2014). Forefire: open-source code for wildland fire spread models. In Domingos, X. V., editor, *Advances in forest fire research*, pages 275–282. Imprensa da Universidade de Coimbra. DOI: 10.14195/978-989-26-0884-6_29.

[Finney, 1998] Finney, M. A. (1998). FARSITE, Fire Area Simulator–model development and evaluation. Research Paper RMRS-RP-4, US Department of Agriculture, Forest Service, Rocky Mountain Research Station.

[FSDeveloper, 2020] FSDeveloper (2020). BGL File Format. FSDeveloper Wiki. Available online at https://www.fsdeveloper.com/wiki/index.php?title=BGL_File_Format. Last seen in 2020-06-09.

[Fujimoto, 2015] Fujimoto, R. M. (2015). Parallel and distributed simulation. In *Proceedings of the 2015 Winter Simulation Conference (WSC '15), December 6–9 2015, Huntington Beach, California, USA*, pages 45–59. DOI: 10.1109/WSC.2015.7408152.

[Fujimoto, 2016] Fujimoto, R. M. (2016). Research challenges in parallel and distributed simulation. *ACM Transactions on Modeling and Computer Simulation*, 26(4):22:1–22:29. DOI: 10.1145/2866577.

[Fullford, 1996] Fullford, D. A. (1996). Distributed Interactive Simulation: Its Past, Present, and Future. In *Proceedings of the 28th Conference on Winter Simulation (WSC '96), December 2006, Coronado, California, USA*, pages 179–185, Washington, DC, USA. IEEE Computer Society. DOI: 10.1145/256562.256601.

[Garau et al., 2017] Garau, M., Celli, G., Ghiani, E., Pilo, F., and Corti, S. (2017). Evaluation of smart grid communication technologies with a co-simulation platform. *IEEE Wireless Communications*, 24(2):42–49. DOI: 10.1109/MWC.2017.1600214.

[Gomes et al., 2017] Gomes, C., Thule, C., Broman, D., Gorm Larsen, P., and Vangheluwe, H. (2017). Co-simulation: State of the art. *arXiv e-prints*, page arXiv:1702.00686.

[Gomes et al., 2018] Gomes, C., Thule, C., Broman, D., Larsen, P. G., and Vangheluwe, H. (2018). Co-simulation: A survey. *ACM Computing Surveys*, 51(3):49:1–49:33. DOI: 10.1145/3179993.

[Hofer and Loper, 1995] Hofer, R. C. and Loper, M. L. (1995). DIS today [Distributed interactive simulation]. *Proceedings of the IEEE*, 83(8):1124–1137. DOI: 10.1109/5.400453.

[Hong et al., 2016] Hong, T., Sun, H., Chen, Y., Taylor-Lange, S. C., and Yan, D. (2016). An occupant behavior modeling tool for co-simulation. *Energy and Buildings*, 117:272 – 281. DOI: 10.1016/j.enbuild.2015.10.033.

[Hudgins and Secondline, 2018] Hudgins, G. and Secondline, J. (2018). Telemetry Applications of TENA and JMETC. In *Proceedings of the 54th Annual International Telemetering Conference (ITC), November 5–8 2018, Glendale, Arizona, USA*. International Foundation for Telemetering.

[IEEE C/SI, 2003] IEEE C/SI (2003). IEEE Recommended Practice for High Level Architecture (HLA) Federation Development and Execution Process (FEDEP). Recommended Practice IEEE 1516.3-2003, IEEE C/SI - Simulation Interoperability Stds Organization/Stds Activity Committee.

[IEEE C/SI, 2010a] IEEE C/SI (2010a). IEEE Recommended Practice for Distributed Simulation Engineering and Execution Process (DSEEP). Recommended Practice IEEE 1730-2010, IEEE C/SI - Simulation Interoperability Stds Organization/Stds Activity Committee.

[IEEE C/SI, 2010b] IEEE C/SI (2010b). IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA)– Framework and Rules. Standard IEEE 1516-2010, IEEE C/SI - Simulation Interoperability Stds Organization/Stds Activity Committee.

[IEEE C/SI, 2012] IEEE C/SI (2012). IEEE Standard for Distributed Interactive Simulation–Application Protocols. Standard IEEE 1278.1-2012, IEEE C/SI - Simulation Interoperability Stds Organization/Stds Activity Committee.

[IEEE C/SI, 2015] IEEE C/SI (2015). IEEE Standard for Distributed Interactive Simulation (DIS) – Communication Services and Profiles. Standard IEEE 1278.2-2015, IEEE C/SI - Simulation Interoperability Stds Organization/Stds Activity Committee.

[Kiranoudis, 2013] Kiranoudis, C. T. (2013). *Forest Fire Simulation User's Manual*. National Technical University of Athens.

[Kosztra et al., 2019] Kosztra, B., Büttner, G., Hazeu, G., and Arnold, S. (2019). Updated CLC illustrated nomenclature guidelines. Service Contract No 3436/R0-Copernicus/EEA.57441, Task 3, D3.1 – Part 1, European Environment Agency, Austria.

[Králíček, 2011] Králíček, T. (2011). Building an air traffic simulation platform using open-source components. Master's thesis, Faculty of Informatics, Masaryk University, Brno, Czech Republic.

[Lasnier et al., 2013] Lasnier, G., Cardoso, J., Siron, P., Pagetti, C., and Derler, P. (2013). Distributed simulation of heterogeneous and real-time systems. In *Proceedings of the 2013 IEEE/ACM 17th International Symposium on Distributed Simulation and Real Time Applications (DS-RT '13), October 30–November 1 2013, Delft, Netherlands*, pages 55–62, Washington, DC, USA. IEEE Computer Society. DOI: 10.1109/DS-RT.2013.14.

[Li et al., 2017] Li, X., Huang, Q., and Wu, D. (2017). Distributed Large-Scale Co-Simulation for IoT-Aided Smart Grid Control. *IEEE Access*, 5:19951–19960. DOI: 10.1109/ACCESS.2017.2753463.

[Luo et al., 2008] Luo, W., Taylor, M. C., and Parker, S. R. (2008). A comparison of spatial interpolation methods to estimate continuous wind speed surfaces using irregularly distributed data from England and Wales. *International Journal of Climatology*, 28(7):947–959. DOI: 10.1002/joc.1583.

[Madden and Glaab, 2017] Madden, M. M. and Glaab, P. C. (2017). Distributed Simulation Using DDS and Cloud Computing. In *Proceedings of the 50th Annual Simulation Symposium (ANSS '17), April 23–26 2017, Virginia Beach, Virginia, USA*, pages 3:1–3:12, San Diego, CA, USA. Society for Computer Simulation International. DOI: 10.5555/3106388.3106391.

[Majdalawieh et al., 2006] Majdalawieh, M., Parisi-Presicce, F., and Wijesekera, D. (2006). DNPSec: Distributed Network Protocol Version 3 (DNP3) Security Framework. In Elleithy, K., Sobh, T., Mahmood, A., Iskander, M., and Karim, M., editors, *Advances in Computer, Information, and Systems Sciences, and Engineering. Proceedings of the 2005 International Joint Conference on Computer, Information, and System Sciences, and Engineering (CISSE 2005), December 10–20 2005*, pages 227–234, Dordrecht. Springer Netherlands. DOI: 10.1007/1-4020-5261-8_36.

[Malvar, 2000] Malvar, H. S. (2000). Fast progressive image coding without wavelets. In *Proceedings of the 2000 Data Compression Conference (DCC 2000), March 28–30 2000, Snowbird, Utah, USA*, pages 243–252. DOI: 10.1109/DCC.2000.838164.

[Manbachi et al., 2016] Manbachi, M., Sadu, A., Farhangi, H., Monti, A., Palizban, A., Ponci, F., and Arzanpour, S. (2016). Real-Time Co-Simulation Platform for Smart Grid Volt-VAR Optimization Using IEC 61850. *IEEE Transactions on Industrial Informatics*, 12(4):1392–1402. DOI: 10.1109/TII.2016.2569586.

[MarketsandMarkets, 2019] MarketsandMarkets (2019). Unmanned Aerial Vehicle (UAV) Market by Vertical, Class, System, Industry (Defense & Security, Agriculture, Construction & Mining, Media & Entertainment), Type, Mode of Operation, Range, Point of Sale, MTOW and Region - Global Forecast to 2025. Technical report, MarketsandMarkets.

[Melman et al., 2015] Melman, S., Pavin, A., Bobkov, V., and Inzartsev, A. (2015). Distributed simulation framework for investigation of autonomous underwater vehicles' real-time behavior. In *Proceedings of the OCEANS 2015 Conference and Exposition, October 19–22 2015, Washington, DC, USA*, pages 1–8. DOI: 10.23919/OCEANS.2015.7404479.

[Microsoft Corporation, 2008a] Microsoft Corporation (2008a). ESP SDK Overview. Microsoft Docs. Available online at https://docs.microsoft.com/en-us/previous-versions/microsoft-esp/cc526948(v=msdn.10). Last seen in 2020-06-09.

[Microsoft Corporation, 2008b] Microsoft Corporation (2008b). SimConnect SDK Reference. Microsoft Docs. Available online at https://docs.microsoft.com/en-us/previous-versions/microsoft-esp/cc526983(v=msdn.10). Last seen in 2020-06-09.

[Microsoft Corporation, 2008c] Microsoft Corporation (2008c). Terrain and Scenery. Microsoft Docs. Available online at https://docs.microsoft.com/en-us/previous-versions/microsoft-esp/cc707102(v=msdn.10). Last seen in 2020-06-09.

[Morakinyo et al., 2017] Morakinyo, T. E., Dahanayake, K., Ng, E., and Chow, C. L. (2017). Temperature and cooling demand reduction by green-roof types in different climates and urban densities: A co-simulation parametric study. *Energy and Buildings*, 145:226–237. DOI: 10.1016/j.enbuild.2017.03.066.

[Nelson, 2001] Nelson, R. M. (2001). Chapter 4 - water relations of forest fuels. In Johnson, E. A. and Miyanishi, K., editors, *Forest Fires*, pages 79–149. Academic Press, San Diego. DOI: 10.1016/B978-012386660-8/50006-4.

[NIMA, 2004] NIMA (2004). Department of Defense World Geodetic System 1984, Its Definition and Relationships With Local Geodetic Systems. Technical Report NIMA TR8350.2, National Imagery and Mapping Agency.

[Noseworthy, 2008] Noseworthy, J. R. (2008). The Test and Training Enabling Architecture (TENA) Supporting the Decentralized Development of Distributed Applications and LVC Simulations. In *Proceedings of the 2008 12th IEEE/ACM International Symposium on Distributed Simulation and Real-Time Applications, October 27–29 2008, Vancouver, BC, Canada*, pages 259–268. DOI: 10.1109/DS-RT.2008.35.

[Okoli and Pawlowski, 2004] Okoli, C. and Pawlowski, S. D. (2004). The Delphi method as a research tool: an example, design considerations and applications. *Information & Management*, 42(1):15–29. DOI: 10.1016/j.im.2003.11.002.

[Olson et al., 2001] Olson, D. M., Dinerstein, E., Wikramanayake, E. D., Burgess, N. D., Powell, G. V. N., Underwood, E. C., D'amico, J. A., Itoua, I., Strand, H. E., Morrison, J. C., Loucks, C. J., Allnutt, T. F., Ricketts, T. H., Kura, Y., Lamoreux, J. F., Wettengel, W. W., Hedao, P., and Kassem, K. R. (2001). Terrestrial Ecoregions of the World: A New Map of Life on Earth: A new global map of terrestrial ecoregions provides an innovative tool for conserving biodiversity. *BioScience*, 51(11):933–938. DOI: 10.1641/0006-3568(2001)051[0933:TEOTWA]2.0.CO;2.

[OMG, 2015] OMG (2015). Data Distribution Service (DDS) – Version 1.4. Standard formal/2015-04-10, Object Management Group.

[OMG, 2018] OMG (2018). DDS Security – Version 1.1. Standard formal/2018-04-01, Object Management Group.

[Pardo-Castellote, 2003] Pardo-Castellote, G. (2003). OMG Data-Distribution Service: architectural overview. In *Proceedings of the 23rd International Conference on Distributed Computing Systems Workshops (ICDCS 2003 Workshops), May 19–22 2003, Providence, Rhode Island, USA*, pages 200–206. DOI: 10.1109/ICDCSW.2003.1203555.

[Rothermel, 1972] Rothermel, R. C. (1972). A mathematical model for predicting fire spread in wildland fuels. Research Paper INT-115, Intermountain Forest & Range Experiment Station, Forest Service, US Department of Agriculture.

[Rothermel, 1991] Rothermel, R. C. (1991). Predicting behavior and size of crown fires in the northern rocky mountains. Research Paper INT-438, US Department of Agriculture, Forest Service, Intermountain Forest and Range Experiment Station.

[Schloegl et al., 2015] Schloegl, F., Rohjans, S., Lehnhoff, S., Velasquez, J., Steinbrink, C., and Palensky, P. (2015). Towards a classification scheme for co-simulation approaches in energy systems. In *Proceedings of the 2015 International Symposium on Smart Electric Distribution Systems and Technologies (EDST), September 8–11 2015, Vienna, Austria*, pages 516–521. DOI: 10.1109/SEDST.2015.7315262.

[Schweiger et al., 2018] Schweiger, G., Engel, G., Schöggl, J. P., Hafner, I., Gomes, C., and Nouidui, T. (2018). Co-simulation – an empirical survey: Applications, recent developments and future challenges. In *Proceedings of the 9th Vienna Conference on Mathematical Modelling (MATHMOD 2018), February 21–23 2018, Vienna, Austria*, pages 125–126. DOI: 10.11128/arep.55.a55286.

[Scott and Reinhardt, 2001] Scott, J. and Reinhardt, E. (2001). Assessing crown fire potential by linking models of surface and crown fire potential. Research Paper RMRS-29, US Department of Agriculture, Forest Service, Rocky Mountain Research Station.

[Silva, 2011] Silva, D. C. (2011). *Cooperative Multi-Robot Missions: Development of a Platform and a Specification Language*. PhD thesis, Faculty of Engineering, University of Porto, Porto, Portugal.

[Silva et al., 2016] Silva, D. C., Abreu, P. H., Reis, L. P., and Oliveira, E. (2016). Development of a Flexible Language for Disturbance Description for Multi-Robot Missions. *Journal of Simulation*, 10(3):166–181. DOI: 10.1057/jos.2015.4.

[Silva et al., 2017] Silva, D. C., Abreu, P. H., Reis, L. P., and Oliveira, E. (2017). Development of flexible languages for scenario and team description in multirobot missions. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, 31(1):69–86. DOI: 10.1017/S0890060416000184.

[Tavella et al., 2016] Tavella, J., Caujolle, M., Vialle, S., Dad, C., Tan, C., Plessis, G., Schumann, M., Cuccuru, A., and Revol, S. (2016). Toward an accurate and fast hybrid multi-simulation with the FMI-CS standard. In *Proceedings of the 2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA), Sep 6–9 2016, Berlin, Germany*, pages 1–5. DOI: 10.1109/ETFA.2016.7733616.

[TC 57, 2020] TC 57 (2020). Communication networks and systems for power utility automation - ALL PARTS. Standard IEC 61850:2020 SER, International Electrotechnical Commission.

[Vecchiola et al., 2008] Vecchiola, C., Grosso, A., and Boccalatte, A. (2008). AgentService: a framework to develop distributed multiagent systems. *International Journal of Agent-Oriented Software Engineering*, 2(3):290–323. DOI: 10.1504/IJAOSE.2008.019421.

[Wagner, 1977] Wagner, C. E. V. (1977). Conditions for the start and spread of crown fire. *Canadian Journal of Forest Research*, 7(1):23–34. DOI: 10.1139/x77-004.

[Wilson et al., 2017] Wilson, J., Schultz, K., Heien, E., Sachs, M., and Rundle, J. (2017). Virtual Quake v3.1.1. DOI: 10.5281/zenodo.1098321.

[Ziv and Lempel, 1977] Ziv, J. and Lempel, A. (1977). A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory*, 23(3):337–343. DOI: 10.1109/TIT.1977.1055714.

Appendix A

Olson Land Classification to CORINE Land Cover Matching

This appendix shows the matching between the different land classification systems that were used in the implemented solution. The matching was done only from the Olson Land Classification system to CLC, and not vice-versa.

Table A.1: Olson Land Classification to CORINE Land Cover Matching

Value	Olson Land Classification	Code	CORINE Land Cover
0	Ocean, Sea, Large Lake	523	Sea and ocean
1	Large City Urban Grid Wet	111	Continuous urban fabric
2	Low Sparse Grassland	321	Natural grasslands
3	Coniferous Forest	312	Coniferous forest
4	Deciduous Conifer Forest	312	Coniferous forest
5	Deciduous Broadleaf Forest	311	Broad-leaved forest
6	Evergreen Broadleaf Forests	311	Broad-leaved forest
7	Tall Grasses And Shrubs	323	Sclerophyllous vegetation
8	Bare Desert	331	Beaches, dunes, sands
9	Upland Tundra	322	Moors and heathland
10	Irrigated Grassland	321	Natural grasslands
11	Semi Desert	331	Beaches, dunes, sands
12	Dry Crop and Town	243	Land principally occupied by agriculture, with significant areas of natural vegetation
13	Wooded Wet Swamp	411	Inland marshes
16	Shrub Evergreen	323	Sclerophyllous vegetation
17	Shrub Deciduous	323	Sclerophyllous vegetation
19	Evergreen Forest And Fields	312	Coniferous forest
20	Cool Rain Forest	313	Mixed forest

21	Conifer Boreal Forest	312	Coniferous forest
22	Cool Conifer Forest	312	Coniferous forest
23	Cool Mixed Forest	313	Mixed forest
24	Mixed Forest	313	Mixed forest
25	Cool Broadleaf Forest	311	Broad-leaved forest
26	Southern Deciduous Broadleaf Forest	311	Broad-leaved forest
27	Conifer Forest	312	Coniferous forest
28	Montane Tropical Forests	313	Mixed forest
29	Seasonal Tropical Forest	313	Mixed forest
30	Cool Crops And Towns	243	Land principally occupied by agriculture, with significant areas of natural vegetation
31	Crops And Town	243	Land principally occupied by agriculture, with significant areas of natural vegetation
32	Dry Tropical Woods	311	Broad-leaved forest
33	Tropical Rainforest	313	Mixed forest
34	Tropical Degraded Forest	313	Mixed forest
35	Corn And Beans Cropland	241	Annual crops (permanent crops)
36	Rice Paddy And Field	213	Rice fields
37	Hot Irrigated Cropland	212	Permanently irrigated land
38	Cool Irrigated Cropland	212	Permanently irrigated land
40	Cool Grasses And Shrubs	322	Moors and heathland
41	Hot And Mild Grasses And Shrubs	321	Natural grasslands
42	Cold Grassland	321	Natural grasslands
43	Savanna (Woods)	324	Transitional woodland-shrub
44	Mire Bog Fen	412	Peat bogs
45	Marsh Wetland	411	Inland marshes
46	Mediterranean Scrub	323	Sclerophyllous vegetation
47	Dry Woody Scrub	324	Transitional woodland-shrub
48	Dry Evergreen Woods	311	Broad-leaved forest
50	Sand Desert	331	Beaches, dunes, sands
51	Semi Desert Shrubs	333	Sparsely vegetated areas
52	Semi Desert Sage	323	Sclerophyllous vegetation
53	Barren Tundra	322	Moors and heathland
54	Cool Southern Hemisphere Mixed Forests	313	Mixed forest
55	Cool Fields And Woods	311	Broad-leaved forest
57	Cool Forest And Field	313	Mixed forest
58	Fields And Woody Savanna	324	Transitional woodland-shrub

59	Succulent And Thorn Scrub	323	Sclerophyllous vegetation
60	Small Leaf Mixed Woods	313	Mixed forest
61	Deciduous And Mixed Boreal Forest	313	Mixed forest
62	Narrow Conifers	312	Coniferous forest
63	Wooded Tundra	324	Transitional woodland-shrub
64	Heath Scrub	322	Moors and heathland
69	Polar And Alpine Desert	335	Glaciers and perpetual snow
72	Mangrove	421	Salt marshes
76	Crop And Water Mixtures	242	Complex cultivation patterns
78	Southern Hemisphere Mixed Forest	313	Mixed forest
89	Moist Eucalyptus	311	Broad-leaved forest
90	Rain Green Tropical Forest	313	Mixed forest
91	Woody Savanna	324	Transitional woodland-shrub
92	Broadleaf Crops	311	Broad-leaved forest
93	Grass Crops	321	Natural grasslands
94	Crops Grass Shrubs	322	Moors and heathland
95	Grass Skirting 1	321	Natural grasslands
96	Grass Skirting 2	321	Natural grasslands
97	Grass and Shrub Skirting	322	Moors and heathland
98	Dry Grass and Dirt Skirting	322	Moors and heathland
99	Sand and Desert Skirting	331	Beaches, dunes, sands
100	Ocean, Sea, Large Lake	523	Sea and ocean
101	Large City Urban Grid Wet	111	Continuous urban fabric
102	Large City Urban Grid Dry	111	Continuous urban fabric
103	Large City Urban Non Grid Wet	111	Continuous urban fabric
104	Large City Urban Non Grid Dry	111	Continuous urban fabric
105	Medium City Urban Grid Wet	111	Continuous urban fabric
106	Medium City Urban Grid Dry	111	Continuous urban fabric
107	Medium City Urban Non Grid Wet	111	Continuous urban fabric
108	Medium City Urban Non Grid Dry	111	Continuous urban fabric
109	Large City Suburban Grid Wet	112	Discontinuous urban fabric
110	Large City Suburban Grid Dry	112	Discontinuous urban fabric
111	Large City Suburban Non Grid Wet	112	Discontinuous urban fabric
112	Large City Suburban Non Grid Dry	112	Discontinuous urban fabric
113	Medium City Suburban Grid Wet	112	Discontinuous urban fabric
114	Medium City Suburban Grid Dry	112	Discontinuous urban fabric
115	Medium City Suburban Non Grid Wet	112	Discontinuous urban fabric
116	Medium City Suburban Non Grid Dry	112	Discontinuous urban fabric
117	Small City Suburban Grid Wet	112	Discontinuous urban fabric
118	Small City Suburban Grid Dry	112	Discontinuous urban fabric

119	Small City Suburban Non Grid Wet	112	Discontinuous urban fabric
120	Small City Suburban Non Grid Dry	112	Discontinuous urban fabric
121	Large City High-rise	111	Continuous urban fabric
122	Ice	335	Glaciers and perpetual snow
123	Inland Water	411	Inland marshes
124	Ocean Inlet	421	Salt marshes
125	Non Perennial Inland Water	411	Inland marshes
126	Non Perennial Inland Sea	411	Inland marshes
127	Reef	332	Bare rock
128	Grass	321	Natural grasslands
129	Arid	333	Sparsely vegetated areas
130	Rock	332	Bare rock
131	Dirt	333	Sparsely vegetated areas
132	Coral	332	Bare rock
133	Lava	332	Bare rock
134	Park	141	Green urban areas
135	Golf Course	142	Sport and leisure facilities
136	Cement	122	Road and rail networks and associated land
137	Tan Sand Beach	331	Beaches, dunes, sands
138	Black Sand Beach	331	Beaches, dunes, sands
139	Airfield1	124	Airports
140	Airfield2	124	Airports
141	Rock Volcanic	332	Bare rock
142	Rock Ice	335	Glaciers and perpetual snow
143	Glacier Ice	335	Glaciers and perpetual snow
144	Evergreen Tree Crop	312	Coniferous forest
145	Deciduous Tree Crop	311	Broad-leaved forest
146	Desert Rock	332	Bare rock
147	Savanna Grass	322	Moors and heathland

Appendix B

Fire Simulation Experiments

This appendix includes the data from the various experiments ran on the implemented solution. These experiments ended after 1 hour of simulation time. Experiment I is the baseline with a randomly selected location and wind speed 15 knots at 135 degrees and can be found in Table B.1. The second experiment uses the same location and wind and was ran to observe the performance variability between executions. Table B.2 shows its data. Experiment III doubles the wind speed of the previous experiments. Its experimental data can be seen in Table B.3. Experiment IV has a bigger domain area than experiments I and II (1 more QMID layer surrounding the original one), but otherwise has the same conditions. Table B.4 details the experiment's data. Finally, experiment V has the same conditions as experiments I and II, but with two fire fronts, and its data can be observed in B.5.

Table B.1: Experiment I Data

Simulated Time (s)	Real Time (s)	Number of Vertexes	Projection Conversion Time
29	3.600048	6	0.117649
7	1.220643	8	0.19133
5	1.216208	9	0.181983
4	0.985353	10	0.189917
18	3.114209	11	0.226168
72	11.15213	12	0.226867
54	8.46984	13	0.25928
14	2.679576	12	0.247758
4	0.86382	13	0.242185
75	11.87271	14	0.481717
35	5.346303	15	0.280006
43	6.668729	16	0.302369
33	5.640187	17	0.444591
12	2.479512	18	0.33166
44	7.096336	19	0.350079

72	11.5264	20	0.37353
80	12.41205	21	0.623276
9	2.145283	22	0.40013
33	5.512438	23	0.422849
74	11.90522	24	0.441005
11	2.224271	25	0.471855
59	9.63848	26	0.640766
6	1.569118	25	0.475913
20	3.327142	26	0.540709
2	0.972013	27	0.518427
10	2.134886	28	0.52109
94	14.81474	29	0.609151
64	10.23667	30	0.582317
3	1.206391	31	0.632936
76	12.55722	32	0.92857
45	7.501552	33	0.877928
14	3.341545	34	0.808016
53	8.945046	35	0.955379
17	3.511785	36	0.715906
59	9.839478	37	0.751866
22	4.503867	38	0.966737
64	10.79207	39	0.765918
102	16.38383	40	0.796074
42	7.122076	41	0.789684
15	3.419923	42	0.794507
53	8.500472	41	0.813273
2	1.40144	42	0.841951
1	0.953864	43	0.847916
44	7.930875	44	1.306701
30	5.818118	45	0.925116
57	9.508793	46	0.905156
49	8.500404	47	0.913559
5	1.526274	48	0.965548
54	9.226433	47	0.938571
8	1.95391	48	0.939938
29	5.62499	49	1.124055
34	6.084584	48	0.981985
8	2.872889	49	1.422807
9	3.137505	50	1.279466
8	2.192791	51	1.027435

46	7.982185	52	1.052561
45	7.89556	53	1.120695
89	14.48223	52	1.038959
11	3.012616	53	1.085886
34	6.851621	54	1.474097
15	3.979228	55	1.117757
79	13.51962	56	1.352701
23	4.925473	57	1.200722
22	5.527473	58	1.949331
1	2.338254	59	1.950626
48	9.426164	60	1.522629
131	21.834	61	1.463882
61	10.84437	60	1.230407
2	1.655388	61	1.22741
28	5.402144	62	1.202088
2	1.508573	63	1.251898
33	6.235259	64	1.276727
1	1.570443	65	1.46608
7	2.34224	66	1.330482
59	10.73994	67	2.137384
2	2.685236	68	2.143092
106	17.38604	69	1.361797
85	14.17162	68	1.325108
12	3.16705	69	1.395961
34	7.443078	70	1.946869
15	4.457137	71	1.732757
27	6.09212	72	1.460163
11	3.086787	73	1.462484
35	7.501792	74	1.793122
53	10.15173	75	1.459335
29	6.543612	76	1.803496
13	4.307995	75	1.850192
13	3.493421	76	1.567887
35	7.074058	77	1.963206
10	3.926359	78	1.587959
1	1.639572	77	1.534392
11	3.132997	78	1.514752
14	3.626717	79	1.549487
4	2.53975	78	1.841143
24	5.832612	79	2.236813

12	3.355014	80	1.58293
27	5.644793	81	1.597342
8	3.180844	82	1.611457
26	5.554272	83	1.610089
16	4.279887	84	1.901677
38	7.826672	85	1.705023
34	7.413086	86	1.726581
38	7.95784	87	1.714899
52	9.993686	88	1.704442
78	14.06548	89	1.820469
28	6.437309	90	1.764901
40	8.463393	92	1.816856
30	6.926502	93	1.903472
24	6.278782	94	2.081307

Table B.2: Experiment II Data

Simulated Time (s)	Real Time (s)	Number of Vertexes	Projection Conversion Time
29	3.60207	6	0.142443
7	1.20884	8	0.174229
5	1.226879	9	0.186275
4	0.992061	10	0.218331
18	3.133281	11	0.263195
72	11.19467	12	0.271246
54	8.473403	13	0.280314
14	2.687803	12	0.249019
4	0.871256	13	0.268405
75	11.69827	14	0.349837
35	5.351369	15	0.337322
43	6.824495	16	0.432533
33	5.591127	17	0.355224
12	2.376358	18	0.369911
44	7.08897	19	0.382645
72	11.54905	20	0.406455
80	12.15779	21	0.436614
9	2.19787	22	0.458072
33	5.56764	23	0.472927
74	11.90525	24	0.489841
11	2.23583	25	0.492884
59	9.541328	26	0.530002

6	1.626378	25	0.62062
20	3.515527	26	0.643929
2	1.088215	27	0.581184
10	2.239843	28	0.557528
94	14.79584	29	0.583627
64	10.29755	30	0.633366
3	1.304105	31	0.664341
76	12.3229	32	0.651077
45	7.402294	33	0.771355
14	3.076792	34	0.680376
53	8.710587	35	0.717638
17	3.473341	36	0.74004
59	10.09858	37	1.02612
22	4.554807	38	0.784538
64	10.90461	39	0.97848
102	16.83511	40	1.075055
42	7.254557	41	0.846558
15	3.424319	42	0.86202
53	8.510081	41	0.825746
2	1.372972	42	0.815843
1	0.951308	43	0.847909
44	7.645225	44	1.019763
30	5.746148	45	0.946996
57	9.509248	46	0.91308
49	8.906627	47	1.168191
5	1.510762	48	0.951095
54	9.221078	47	0.92635
8	1.957758	48	0.942153
29	5.579664	49	1.078985
34	6.385875	48	1.280011
8	2.789539	49	1.17419
9	2.748142	50	1.05225
8	2.237925	51	1.073761
46	8.063708	52	1.137601
45	7.957199	53	1.180396
89	14.53012	52	1.081148
11	3.010269	53	1.120724
34	6.881232	54	1.446444
15	3.869511	55	1.164747
79	13.49374	56	1.365266

23	4.956237	57	1.302161
22	5.554451	58	1.911152
1	2.3432	59	2.061782
48	9.474654	60	1.431554
131	21.579	61	1.348448
61	10.82506	60	1.241789
2	1.480123	61	1.191522
28	5.471714	62	1.272591
2	1.547485	63	1.291104
33	6.314199	64	1.354156
1	1.588943	65	1.485537
7	2.316003	66	1.3019
59	10.60069	67	2.006063
2	2.720191	68	2.18581
106	17.47951	69	1.451755
85	14.20058	68	1.355882
12	3.144987	69	1.372722
34	7.167466	70	1.627221
15	4.213923	71	1.69474
27	6.020337	72	1.425292
11	3.129247	73	1.508994
35	7.537629	74	1.866068
53	10.16798	75	1.582373
29	6.589245	76	1.870586
13	4.251543	75	1.732829
13	3.58172	76	1.659356
35	7.077217	77	1.971684
10	3.950269	78	1.663379
1	1.649121	77	1.544687
11	3.204286	78	1.583579
14	3.859988	79	1.785801
4	2.395535	78	1.599539
24	5.741184	79	2.150189
12	3.402423	80	1.62727
27	5.759061	81	1.709618
8	3.369007	82	1.652524
26	5.658377	83	1.652114
16	4.174	84	1.79711
38	7.812358	85	1.726373
34	7.346849	86	1.693642

38	8.065519	87	1.847189
52	10.07618	88	1.729202
78	14.02317	89	1.761624
28	6.526953	90	1.922802
40	8.538067	92	1.970981
30	6.88222	93	1.827228
24	6.143149	94	1.950035

Table B.3: Experiment III Data

Simulated Time (s)	Real Time (s)	Number of Vertexes	Projection Conversion Time
22	2.277519	6	0.187223
2	1.519922	8	0.166822
1	0.505119	9	0.205131
14	2.011973	10	0.209219
5	1.310718	11	0.248051
22	3.593158	12	0.261319
23	3.762407	13	0.268806
47	7.265902	14	0.282305
8	1.677065	15	0.316008
13	2.342214	16	0.329074
19	3.101435	17	0.341815
16	2.985824	18	0.365062
11	2.026584	17	0.340248
4	1.040692	18	0.378129
12	2.391776	19	0.544426
29	4.946401	20	0.402514
11	2.186172	21	0.426082
38	6.302032	22	0.45462
35	5.904484	23	0.54962
15	2.833104	24	0.488709
23	4.105581	25	0.505462
7	1.684255	26	0.538425
37	6.445187	27	0.912103
1	1.40323	28	1.147495
43	7.519262	29	0.589437
18	3.409187	30	0.611019
5	1.477057	31	0.634755
8	2.007517	32	0.656103
32	5.874337	33	0.929578

6	1.963735	34	0.879922
7	1.717046	35	0.704415
62	10.33971	35	1.136662
1	1.393219	36	1.170142
22	4.054652	37	0.767957
4	1.546307	38	0.78072
29	5.143394	39	0.789511
5	1.64547	40	0.807727
7	1.92365	41	0.833426
23	4.469144	42	0.837731
11	2.667587	44	1.047792
16	3.916372	43	1.441088
1	1.757553	45	1.466422
3	1.356323	46	0.944689
28	5.416427	47	0.946474
46	8.145218	48	1.218676
6	2.737608	47	1.56746
1	2.159559	48	1.775974
7	2.490471	49	1.000166
20	4.178648	50	1.031184
1	1.116754	51	1.012807
3	1.422212	52	1.015774
9	2.519688	53	1.198589
31	6.109235	55	1.091345
12	3.167032	56	1.11964
67	11.69464	57	1.199408
34	6.893262	58	1.904367
7	2.673938	59	2.323746
6	3.096275	60	1.566169
38	7.632529	61	1.367307
9	2.97678	60	1.201183
22	4.87943	61	1.677487
3	2.444721	62	1.702935
34	8.13886	63	2.725859
1	3.151341	64	2.816403
11	4.271883	65	1.287685
7	2.348363	66	1.335116
21	4.531982	67	1.393786
27	5.73785	68	1.331718
13	3.761208	69	1.464565

31	6.707375	70	2.27209
2	2.832284	69	2.237777
2	1.823155	70	1.411806
6	2.478644	71	1.398659
27	5.511768	72	1.463124
25	5.571146	73	1.473492
36	7.413917	74	1.522731
10	3.996985	75	2.163349
13	4.930123	76	2.596508
3	3.274812	77	2.440956
53	10.48228	76	2.465399
11	3.732215	77	3.107791
8	4.493649	78	2.440225
14	6.40544	79	3.581018
1	3.466564	80	3.111495
30	6.126248	81	1.629655
37	7.776813	82	2.157528
3	2.052762	83	1.641437
1	1.776195	84	1.637016
18	4.372921	85	1.692125
30	7.158254	86	1.996627
30	7.234047	88	3.789893
7	4.668749	89	4.195837
15	5.965078	90	4.590394
10	5.741246	91	3.303331
67	13.97183	90	2.987298
3	3.575699	91	2.943233
13	4.09281	92	2.142027
19	6.394505	91	3.494362
4	5.088756	92	4.357047
1	4.668655	91	3.692581
7	2.998527	92	1.983418
5	3.033406	91	2.319143
5	5.05154	93	4.789319
19	8.227612	92	5.18421
4	5.414909	94	4.308612
3	2.974468	95	2.716669
1	3.13561	94	2.827154
4	2.494373	95	1.926163
33	5.844926	96	2.254547

9	4.206478	97	2.159939
32	7.169236	98	2.525823
6	3.804272	97	2.406792
21	6.1	98	1.953025
20	6.337015	99	4.36335
17	7.274537	101	6.713934
15	7.656865	102	5.2405
6	6.05828	103	4.235981
37	8.96623	104	2.664185
7	4.118725	105	2.527505
26	7.07815	104	3.266998
6	4.651277	103	3.031361
12	5.044355	104	2.077675
17	6.384249	103	3.37528
14	4.173797	104	3.802069
19	6.496166	105	3.017389
11	5.972078	106	4.675186
18	8.365229	107	6.672229
13	10.44454	108	7.369376
3	7.866419	109	7.489595
2	7.450784	110	6.438306
19	8.355604	111	2.203694
19	6.471585	110	3.420214
3	4.887832	111	4.035487
33	8.122599	112	2.872724
9	4.621487	113	2.564399
39	8.346757	114	2.716532
7	4.736048	115	3.070714
18	6.870126	116	2.967557
48	10.51156	117	4.621442
16	7.26447	118	5.159806
20	6.723511	119	4.525984
12	6.502928	120	3.427432
41	10.2627	121	3.41797
4	4.55229	120	3.40727
14	7.181173	121	3.943873
18	5.024041	122	4.262785
15	7.049579	123	3.53132
37	9.392745	124	2.500204
17	5.897477	125	2.50967

52	11.18153	126	2.537473
31	9.415391	127	5.355189
17	6.064434	128	5.568484
15	9.294082	129	7.077868
6	7.062917	130	4.576019
53	12.46911	131	2.524193
24	6.998233	132	2.604883
28	8.345206	134	3.217308
31	8.37074	135	2.741622
13	5.320892	136	2.640693
54	12.00662	136	3.07852
24	6.297419	136	3.230614
13	8.273183	137	5.516917
1	5.9292	138	5.437619
24	8.569978	139	2.904488
15	6.292147	140	3.197278
15	6.483571	141	3.132722
33	11.5439	142	5.797589
2	6.482302	143	5.916983
17	10.36526	142	5.840368
2	6.365724	141	5.821303
3	3.301239	140	2.889976
2	3.565114	141	3.258276
12	5.364973	142	4.348071
6	5.882773	143	4.366925
7	4.220512	144	2.897372
1	3.163525	145	2.96565
22	6.436787	146	3.079145
30	6.340291	147	3.352824
17	9.83844	148	7.453688
13	8.322499	147	7.182062
24	13.04398	148	11.26847
20	12.25822	150	8.13755
14	11.24966	151	8.73462
4	9.230601	152	7.651094
63	18.31879	153	5.840606
2	6.485548	154	5.944062
70	17.54768	153	6.041719
21	11.48005	154	9.66821
30	12.47057	155	10.78566

24	13.14261	156	10.50313
30	14.40065	157	8.996805
16	9.944333	158	8.98995
25	10.28118	161	9.296791
26	13.18605	160	12.20861
38	18.67144	159	13.41397
24	13.35181	162	5.366503
69	16.04179	164	3.349083
65	14.11902	163	4.899759

Table B.4: Experiment IV Data

Simulated Time (s)	Real Time (s)	Number of Vertexes	Projection Conversion Time
29	3.606432	6	0.126339
7	1.191428	8	0.16227
5	1.250698	9	0.197138
4	0.970116	10	0.205119
18	3.099279	11	0.237245
72	11.1542	12	0.229221
54	8.442381	13	0.246422
14	2.805723	12	0.333649
4	0.980084	13	0.340317
75	11.69473	14	0.277995
34	5.324839	15	0.335125
44	6.71006	16	0.303194
33	5.559766	17	0.317108
12	2.529465	18	0.50138
44	7.196838	19	0.354119
72	11.55107	20	0.440767
76	12.06235	21	0.404681
13	2.457516	22	0.540473
33	5.513686	23	0.445888
74	12.07552	24	0.679056
9	2.078989	25	0.540729
61	9.8291	26	0.46156
8	1.62978	25	0.453886
16	3.137676	26	0.525842
4	1.128066	27	0.568223
8	2.219903	28	0.685217
94	15.09963	29	0.703964

65	10.37037	30	0.714715
1	1.186143	31	0.808399
77	12.22537	32	0.592044
44	7.23488	33	0.608964
16	3.189065	34	0.609743
53	8.611732	35	0.619854
17	3.536037	36	0.790298
59	9.795483	37	0.695863
22	4.291109	38	0.755116
62	10.27683	39	0.73175
104	16.63651	40	0.719233
42	7.103117	41	0.731056
16	3.368192	42	0.737443
44	7.695726	41	0.913381
5	1.464668	42	0.7517
7	1.629138	43	0.767775
41	7.124912	44	0.799545
36	6.558521	45	0.991025
54	9.004455	46	0.858184
46	8.050705	47	0.828809
9	2.011124	48	0.8456
46	7.898229	47	0.823525
6	1.834943	48	0.972931
36	6.267224	49	0.84849
30	5.395722	50	0.891799
6	2.042454	51	0.914417
8	2.402187	50	0.880378
26	5.050518	51	1.153402
65	10.7803	52	0.970271
20	3.902031	53	0.917598
69	11.36721	54	0.954268
62	10.59769	55	0.986475
57	9.883225	56	1.004401
21	4.49138	55	0.985439
33	6.309016	56	1.001894
8	2.466689	57	1.022905
13	3.262104	58	1.025765
29	5.992335	59	1.215992
36	6.730217	60	1.088407
6	2.057091	59	1.042505

5	2.243919	60	1.533498
77	12.70894	61	1.228785
34	6.509433	62	1.248368
132	21.38288	63	1.411937
15	3.938389	64	1.308179
40	7.737457	65	1.336877
12	3.896103	66	1.529998
99	17.0334	67	1.507396
38	7.498799	68	1.387129
67	12.37805	69	1.714388
43	8.446872	70	1.473747
17	4.508228	71	1.450125
35	7.494816	72	1.881951
11	4.035192	71	1.776936
3	2.098235	72	1.691565
16	3.84484	73	1.46315
1	1.861994	74	1.757118
24	5.105799	75	1.515775
31	6.386235	76	1.733345
5	2.262198	77	1.550516
20	4.516502	76	1.531315
7	2.7768	77	1.761274
6	2.443692	78	1.578741
42	7.963468	79	1.643967
27	6.0311	78	1.603921
14	3.97626	77	1.551122
9	3.251686	78	1.569672
9	3.717939	79	1.937425
15	4.630583	80	1.741236
49	8.745027	79	1.970793
8	4.029602	80	2.969751
15	4.50196	81	3.467925
11	6.341358	82	3.685662
1	4.101904	83	3.527163
56	10.28104	84	1.833899
8	3.695281	85	1.960878
62	11.77938	86	1.741036
15	4.906605	87	2.188867
16	5.036672	88	2.10579
273	44.15064	89	2.426421

Table B.5: Experiment V Data

Simulated Time (s)	Real Time (s)	Number of Vertexes	Projection Conversion Time
29	3.812694	12	0.278453
7	1.46226	16	0.354361
5	1.490798	18	0.40562
4	1.238178	20	0.438413
18	3.862486	22	0.934068
72	11.8218	24	0.496231
54	8.79529	26	0.543483
14	3.003121	24	0.489005
4	1.221146	26	0.546704
75	12.11631	28	0.659848
35	5.761412	30	0.654531
43	7.407789	32	0.919012
33	6.198238	34	0.736277
12	2.861592	36	0.761701
44	7.632124	38	0.795234
72	12.0459	40	0.818066
80	12.66461	42	0.857421
8	2.804042	44	0.965316
34	6.140758	46	0.923932
74	12.50805	48	0.951497
11	2.96094	50	1.05744
59	10.21405	52	1.047947
6	2.221284	50	1.03387
20	4.506785	52	1.570476
2	2.175727	54	1.506099
10	3.113435	56	1.119793
94	15.66161	58	1.151081
67	11.31717	60	1.657278
3	2.41197	62	1.718271
76	13.31746	64	1.41555
44	8.27948	66	1.649082
15	4.06818	68	1.360367
53	10.43305	69	2.593613
1	2.968513	70	2.657523
17	4.971976	72	1.457398
59	11.49056	74	2.273291
22	5.780455	75	1.570847

1	1.613232	76	1.508876
63	11.12616	78	1.617876
102	17.66201	80	1.751041
42	10.32432	81	3.781983
2	4.389943	82	4.070059
15	5.859138	84	2.372119
56	10.85561	82	3.170953
3	5.553801	83	4.63675
1	4.913626	85	4.390278
1	2.636911	86	2.532273
43	8.431074	88	1.955899
29	7.349533	89	2.389037
1	1.877472	90	1.772973
57	10.40538	92	1.808456
53	13.63592	93	6.18007
4	6.468743	94	6.000149
3	6.316249	96	5.017249
63	11.2133	94	2.916304
8	5.6965	95	4.085067
1	4.378975	96	3.816755
30	9.291673	97	3.616717
1	3.979231	98	3.703449
33	9.183843	97	3.995987
2	4.323228	96	3.756395
8	3.279243	97	2.111414
15	4.783934	98	3.769477
8	5.056725	99	4.470338
6	5.053703	100	3.682393
2	2.202581	101	1.946154
16	4.509549	102	2.099278
30	8.307408	104	3.150794
46	11.99825	105	4.199146
1	4.543151	106	4.206149
100	18.851	108	2.095362
33	8.152947	107	2.672444
21	6.228754	108	3.291534
15	5.817227	109	4.143256
8	5.058013	110	3.249911
1	5.103701	112	4.522669
65	15.65977	111	5.132431

23	8.661432	111	4.097565
6	6.294925	112	5.196248
1	5.458709	113	5.091685
20	7.329275	114	2.364487
1	3.3348	115	3.027351
5	2.951899	116	2.23759
2	2.7037	117	2.44635
6	4.032036	118	2.949216
36	8.655035	120	2.53809
138	23.42769	121	3.605053
7	5.381045	122	3.464123
62	15.46533	121	6.311732
18	9.007249	120	7.765618
14	9.081859	121	7.602251
20	9.934736	122	7.214202
26	10.03523	123	7.301519
19	9.845663	125	7.149854
28	8.858875	126	5.689447
27	8.352586	127	3.271912
10	5.363474	128	2.867366
3	2.936847	129	2.513986
14	4.874903	130	2.498865
27	7.810856	129	6.187808
16	9.57308	130	7.831229
26	11.7405	131	9.122597
26	10.2824	132	8.975539
26	15.07432	133	9.458767
10	13.34584	134	10.79989
3	10.70965	135	10.01968
2	10.79571	136	9.988664
107	27.43929	137	6.857479
3	7.615083	138	6.918297
45	15.62341	137	6.013999
2	6.590847	138	6.039355
38	13.91394	137	6.398143
4	7.026426	138	5.723986
49	12.03168	139	3.443718
19	8.18496	140	6.297503
10	8.000843	141	5.796735
46	12.36114	143	5.331129

18	14.88437	143	12.26992
4	12.1937	145	11.73375
8	14.55982	144	13.02572
3	12.93325	146	11.33217
45	14.57669	147	3.364547
39	13.75281	146	10.73685
27	11.40257	147	10.6988
25	11.81913	148	11.29871
39	16.38293	149	8.333668
28	10.09681	150	8.830385
30	14.85757	151	10.52412
36	14.58097	152	11.10681
25	13.23291	153	12.27826
39	17.23972	152	9.325296
23	11.35128	151	10.12217
44	15.31699	153	11.18708