Bachelor Degree Thesis
Eloi Martín Coll

Flight Plan Information Exchange using SWIM

supervised by
Antoni Broquetas Ibars
Daniel Martínez Farnés

In partial fulfillment of the requirements for the degree in Telecommunication Systems Engineering

Barcelona, January 23, 2018
Abstract

The current need of awareness of the airspace and agility to deliver ATM (Air Traffic Management) information has led to a new concept named SWIM (System Wide Information Management). This Eurocontrol program provides interoperability and standardization along with a net-centric architecture to ensure the efficient exchange of regulated ATM information.

Meanwhile, a UTM (UAS Traffic Management) system is being developed by PildoLabs to coordinate unmanned aircraft for emergency operations. The system needs an external source of ATM information, at the same time that should be able to provide data to third parties. Moreover, the company is working on a real-time flight tracking application called MergeStrip to improve air controller’s task.

In this project is presented a deeper view of SWIM and the technology involved. After testing an actual ATM web service, a SWIM web service has been designed and developed to include a flight plan exchange feature into the UTM system. Finally, it has been integrated into MergeStrip and presented visual results to expose SWIM effectiveness.
Resumen

La necesidad actual de conocimiento del espacio aéreo y la agilidad de distribución de información ATM (de Gestión de Tráfico Aéreo) ha conducido a un nuevo concepto de nombre SWIM (Información de Gestión de Todo el Sistema).

Mientras tanto, un sistema UTM (de Gestión de Tráfico de UAS) está siendo desarrollado por PildoLabs para coordinar aeronaves no tripuladas para operaciones de emergencia. El sistema necesita una fuente externa de información ATM, al mismo tiempo que debería ser capaz de proporcionar datos a terceros. Además, la empresa está trabajando en una aplicación de seguimiento de vuelos en tiempo real llamada MergeStrip para mejorar la labor de los controladores aéreos.

En este proyecto se presenta una visión profunda de SWIM y la tecnología involucrada. Después de examinar un servicio web ATM actual, un servicio web SWIM se ha diseñado y desarrollado para incluir la funcionalidad de intercambio de planes de vuelo en el sistema UTM. Finalmente, se ha integrado en MereStrip y se han presentado resultados visuales para exponer la efectividad de SWIM.
Resum

La necessitat actual de coneixament de l’espai aeri i l’agilitat de distribució d’informació ATM (de Gestió de Trànsit Aeri) ha conduït a un nou concepte de nom SWIM (Informació de Gestió de Tot el Sistema).

Mentrestant, un sistema UTM (de Gestió de Trànsit de UAS) està sent desenvolupat per PildoLabs per coordinar aeronaus no tripulades per operacions d’emergència. El sistema necessita una font externa d’informació ATM, al mateix temps que hauria de ser capaç de proporcionar dades a tercers. A més, l’empresa està treballant en una aplicació de seguiment de vols en temps real anomenada MergeStrip per millorar la labor dels controladors aers.

En aquest projecte es presenta una visió profunda de SWIM i la tecnologia involucrada. Després d’examinar un servei web ATM actual, un servei web SWIM s’ha dissenyat i desenvolupat per incloure la funcionalitat d’intercanvi de plans de vol en el sistema UTM. Finalment, s’ha integrat en MergeStrip i s’han presentat resultats visuals per exposar l’efectivitat de SWIM.
I would like to show my gratitude to Daniel Martínez, my project supervisor in PildoLabs, for his help and support, suggestions and the confidence he placed in me to fulfill this project. His knowledge in programming and his project management experience has helped me progress when struggling with specific issues.

Furthermore, I would also like to take a moment to appreciate the opportunity given by the company. They adapted themselves to my situation and were able to come up with a bachelor degree thesis proposal.

I would also like to thank Prof. Antoni Broquetas, my academic supervisor, for his empathy and effort to accept my unexpected project request.

Last but not least, I want to thank my family and my beloved partner, along with my friends for their encouragement all through this project.
### Revision History and Approval Record

<table>
<thead>
<tr>
<th>Revision</th>
<th>Date</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>10/12/2017</td>
<td>Document Creation</td>
</tr>
<tr>
<td>1</td>
<td>21/12/2017</td>
<td>Document Revision</td>
</tr>
<tr>
<td>2</td>
<td>08/01/2018</td>
<td>Document Revision</td>
</tr>
<tr>
<td>3</td>
<td>20/01/2018</td>
<td>Document Revision</td>
</tr>
<tr>
<td>4</td>
<td>24/01/2018</td>
<td>Document Revision</td>
</tr>
</tbody>
</table>

### Document Distribution List

<table>
<thead>
<tr>
<th>Name</th>
<th>Email</th>
</tr>
</thead>
<tbody>
<tr>
<td>Eloi Martín Coll</td>
<td><a href="mailto:eloi.martin@alu-etsetb.upc.edu">eloi.martin@alu-etsetb.upc.edu</a></td>
</tr>
<tr>
<td>Antoni Broquetas Ibars</td>
<td><a href="mailto:broquetas@tsc.upc.edu">broquetas@tsc.upc.edu</a></td>
</tr>
<tr>
<td>Daniel Martínez Farnés</td>
<td><a href="mailto:daniel.martinez@pildo.com">daniel.martinez@pildo.com</a></td>
</tr>
</tbody>
</table>

### Written by: Reviewed and approved by:

<table>
<thead>
<tr>
<th>Date</th>
<th>Name</th>
<th>Date</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>23/01/2018</td>
<td>Eloi Martín Coll</td>
<td>24/01/2018</td>
<td>Antoni Broquetas Ibars</td>
</tr>
<tr>
<td></td>
<td>Project author</td>
<td></td>
<td>Daniel Martínez Farnés</td>
</tr>
<tr>
<td></td>
<td>Position</td>
<td></td>
<td>Project supervisor</td>
</tr>
</tbody>
</table>
# Contents

1 Introduction ............................................. 1  
   1.1 General Overview .................................... 1  
   1.2 Project Goals ..................................... 2  
   1.3 Requirements and Specifications .................... 2  
   1.4 Work Plan ........................................ 3  

2 State of the Art ........................................ 4  
   2.1 SWIM, Connecting the ATM World .................... 4  
   2.2 AIRM and Exchange Models ......................... 6  
   2.3 SWIM Registry .................................... 8  
   2.4 Web Services ..................................... 9  

3 Development ........................................... 11  
   3.1 Testing ATM Web Services ......................... 11  
   3.2 Infrastructure Design ............................. 13  
   3.3 Web Service Deployment .......................... 14  

4 Results ................................................. 20  
   4.1 Server Validation with SoapUI .................... 20  
   4.2 Client Check using Eclipse ....................... 20  
   4.3 Interoperability Test with Python ................ 22  
   4.4 Web Service Evaluation from MergeStrip .......... 23  

5 Budget .................................................. 25  

6 Conclusions and Future Work ........................... 26  

Bibliography ............................................ 27  

Acronyms ............................................... 29
List of Figures

1.1 Project scenario ............................................. 2
1.2 Project work packages scheduled .......................... 3
1.3 Definitive Gantt chart proposed ........................... 3

2.1 ATM information sharing evolution diagram with SWIM .... 4
2.2 Net-centric information viewpoint .......................... 5
2.3 SWIM sharing information architecture ...................... 6
2.4 Interaction between the service stakeholders and the Registry .... 8
2.5 Web services architecture ..................................... 9

3.1 Python dictionary values for the SOAP request ............... 11
3.2 REST access example using HTTP request .................... 12
3.3 Jumpstart capture of FlightAware tracked flights ............... 13
3.4 Project infrastructure design schema ......................... 14
3.5 Top down web service build .................................. 16
3.6 Java database connection ..................................... 16
3.7 Server accessibility and available services list ................. 17
3.8 Distributed object communication ............................ 18
3.9 Service infrastructure result .................................. 19

4.1 SoapUI $fileNewFlightPlan$ TestCase .......................... 21
4.2 Eclipse $queryFlightsByKeys$ client result .................... 21
4.3 Python client $queryFlightPlans$ response on terminal .......... 22
4.4 Eclipse console with the flight plans received ................ 23
4.5 MergeStrip’s display of the arrivals .......................... 24
Chapter 1

Introduction

This chapter provides a general overview of the project, the main goals to be achieved and the required specifications for the development and the implementation. It also includes a general work plan showing the project’s organization.

1.1 General Overview

The ongoing increase of flights is leading an evolution of the airspace, decreasing its capacity and difficulty the predictability. The current situation require the introduction of a new concept of ATM (Air Traffic Management) information exchange. The latest Eurocontrol program, SWIM (System Wide Information Management), facilitates the data sharing providing an architecture to increase awareness and improve agility to deliver information.

In the same line, the recent presence of UAS (Unmanned Aircraft Systems) relies on a separate ecosystem but complementary to ATM for low-altitude uncontrolled operations. UTM (UAS Traffic Management) includes airspace design, dynamic geofencing, congestion management and terrain avoidance to ensure safe and efficient operations for this newest entrant into the skies.

PildoLabs is developing a UTM system to coordinate unmanned aircraft for emergency operations. To manage these operations, an accurate state of the air traffic is needed, not only for PildoLabs, but also for third parties that need to be aware of the operations. Therefore, in this project is proposed the use of SWIM to incorporate the exchange of ATM information to the system. In particular, the following features will be added: an external source of flight plans and a flight plan provider to third parties.

On the other hand, MergeStrip is an air controller java-based application being also developed by PildoLabs. It offers real-time aircraft tracking around an specific airport, focused on assisting the progress of air controllers task. The features developed in this project fits MergeStrip’s goal and are expected to improve its performance. At the same time, it will provide a well-organized display to show the project results and expose the SWIM concept potential.

The purpose of this thesis is to track the whole project, from the study of the SWIM program and its standards, to the implementation and evaluation of the features developed.
1.2 Project Goals

The project main goals are:

1. Get a comprehensive overall picture of the SWIM program
2. Design and develop a flight plan information exchange infrastructure
3. Integrate the features into MergeStrip
4. Validate and evaluate the developed system

1.3 Requirements and Specifications

Requirements

– The system must manage interoperable information
– Eurocontrol has to successfully deliver flight plans to the system
– Third parties should be able to query for stored flight plans
– An integration into MergeStrip projects is required

Specifications

– The information exchanged should follow the ATM Information Reference Model
– It has to consist on a web server module and a web service client
– The server should contain or access a database
– The client needs to be a java project
1.4 Work Plan

As stated previously, the aim of this section is to provide the most accurate possible project organization to follow.

It has been divided into 6 main work packages, each one with its own characteristic sub-tasks, whose amount of work has been individually evaluated to finally set a proportionally estimated dedication time, as represented in the figures 1.1 and 1.2.

<table>
<thead>
<tr>
<th>Name</th>
<th>Begin date</th>
<th>End date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
<td>9/8/17</td>
<td>9/13/17</td>
</tr>
<tr>
<td>SWIM Program Overview</td>
<td>9/8/17</td>
<td>9/13/17</td>
</tr>
<tr>
<td>ATM Information Reference Model Overview</td>
<td>9/13/17</td>
<td>9/18/17</td>
</tr>
<tr>
<td>Project Planning</td>
<td>9/20/17</td>
<td>9/25/17</td>
</tr>
<tr>
<td>Main Objectives Definition</td>
<td>9/20/17</td>
<td>9/25/17</td>
</tr>
<tr>
<td>Work Plan Draft</td>
<td>9/25/17</td>
<td>9/28/17</td>
</tr>
<tr>
<td>State of the Art</td>
<td>9/29/17</td>
<td>10/2/17</td>
</tr>
<tr>
<td>Global Exchange Models Study</td>
<td>9/29/17</td>
<td>10/11/17</td>
</tr>
<tr>
<td>Jumpstart Test</td>
<td>10/9/17</td>
<td>10/17/17</td>
</tr>
<tr>
<td>Eurocontrol Services Identifiation</td>
<td>10/16/17</td>
<td>10/20/17</td>
</tr>
<tr>
<td>System Development</td>
<td>10/23/17</td>
<td>11/29/17</td>
</tr>
<tr>
<td>Infrastructure Design</td>
<td>10/29/17</td>
<td>10/31/17</td>
</tr>
<tr>
<td>Web Server Module Implementation</td>
<td>10/30/17</td>
<td>11/21/17</td>
</tr>
<tr>
<td>Web Service Client Development</td>
<td>11/3/17</td>
<td>11/28/17</td>
</tr>
<tr>
<td>Review and Improvements</td>
<td>11/10/17</td>
<td>12/21/17</td>
</tr>
<tr>
<td>Strengths and Weaknesses</td>
<td>11/29/17</td>
<td>12/1/17</td>
</tr>
<tr>
<td>Improvements</td>
<td>12/4/17</td>
<td>12/15/17</td>
</tr>
<tr>
<td>Conclusions</td>
<td>12/18/17</td>
<td>12/21/17</td>
</tr>
<tr>
<td>Final Report and Lecture</td>
<td>12/4/17</td>
<td>2/2/18</td>
</tr>
<tr>
<td>Final Report Writing</td>
<td>12/4/17</td>
<td>1/22/18</td>
</tr>
<tr>
<td>Lecture Preparation</td>
<td>1/23/18</td>
<td>2/2/18</td>
</tr>
</tbody>
</table>

Figure 1.2: Project work packages scheduled

Figure 1.3: Definitive Gantt chart proposed

Regarding the fulfillment of the calendar proposed, it must be specified that it has been successfully followed as planned in Project Critical Review, created on the 22-11-2017 [1].
Chapter 2
State of the Art

2.1 SWIM, Connecting the ATM World

The SWIM (System Wide Information Management) program includes a full modification in the concept of how information is managed across its complete lifecycle and along the whole European ATM system. Assuring the provision of commonly understood quality information is the main benefit of the implementation of the SWIM concept. One single solution is obviously not expected and surely not one single technology will rule. However, it has been stated that global interoperability and standardization are necessary and SWIM is playing an important role.

2.1.1 Current Situation and Need for Change

Current ATM system consist of a large range of different applications that has been developed continuously. It also covers custom communication protocols, each one with their own information systems. Furthermore, how the ATM information is structured, defined and provided is peculiar for most of the systems.

The expected increase in aviation capacity demands and attention to the environmental impact are relying ever more on accurate and timely information. Such information must be organized and provided with flexibility, interoperability and security. It has been demonstrated that SWIM results in a more time and cost efficient exchange of information between providers and users. SWIM is still currently being fully deployed in Europe.

Figure 2.1: ATM information sharing evolution diagram with SWIM
2.1.2 SWIM Principles and Design

The aim of SWIM is to provide information users with relevant and commonly understandable information. This information should be of the right quality, provided at the right time and delivered to the right place, so enabling the concept of net-centric ATM operations.

![Net-centric information viewpoint](image)

Figure 2.2: Net-centric information viewpoint

Who needs to share information?

- Pilots: taking off, navigating and landing the aircraft
- Airport Operations Centers: managing departures, surface movements, gates and arrivals
- Airline Operations Centers: building schedules, planning flight routing and fuel uplift, ensuring passenger connections and minimizing the impact of delays
- Air Navigation Service Providers (ANSPs): organizing and managing the airspace over a country and managing air traffic passing through their airspace
- Meteorology Service Providers: providing weather reports and forecasts
- Military Operations Centers: planning missions, blocking airspace to conduct training operations, fulfilling national security tasks

What kind of information needs to be shared?

- Aeronautical: Information resulting from the assembly, analysis and formatting of aeronautical data
- Flight trajectory: the detailed route of the aircraft defined in four dimensions (4D), so that the position of the aircraft is also defined with respect to the time component
- Aerodrome operations: the status of different aspects of the airport, including approaches, runways, taxiways, gate and aircraft turn-around information
- Meteorological: information on the past, current and future state of earth’s atmosphere relevant for air traffic
- Air traffic flow: the network management information necessary to understand the overall air traffic and air traffic services situation

- Surveillance: positioning information from radar, satellite navigation systems, aircraft data links, etc

- Capacity and demand: information on the airspace users needs of services, access to airspace and airports and the aircraft already using it

As seen, the information is provided by a large number of different participants and is made available to a wide range of ATM information consumers. Therefore, the information needs to be clearly and uniquely defined and well understood. In other words, there is a need for semantic interoperability. This requires a detailed definition of the information at a conceptual level and at a level of exchanged data between systems. Satisfying this need is the role of the ATM Information Reference Model (AIRM).

### 2.2 AIRM and Exchange Models

The AIRM is one of the key SWIM standards and thus a key enabler of the SWIM concept. By providing semantic interoperability, it ensures that the meaning of information is not lost or altered as it travels through SWIM. This, in addition, helps to reduce the implementation costs of the information sharing environment.

On a practical level, the AIRM promotes a definition of all ATM information through harmonized conceptual and logical data models such as AIXM, FIXM and WXXM. It contains well-known elements such as Aerodrome, ATS Route, Airspace, Flight procedure and a common definition of fundamental modeling concepts including time and geometry.

![Figure 2.3: SWIM sharing information architecture](image-url)
2.2.1 FIXM

The Flight Information Exchange Model (FIXM) is an exchange model capturing Flight and Flow information that is globally standardized. It is the equivalent for the Flight domain of AIXM (Aeronautical Information Exchange Model) and WXXM (Weather Information Exchange Model), depending on each other to complete definitions where necessary.

The Flight Information Exchange Model (FIXM) is a specification divided into two components:

- A logical information model expressed in UML

- A data exchange format using XML Schema (XSD) technology

The FIXM XML Schemas are a data encoding specification for flight data. They are an implementation of the FIXM UML Model as an XML (Extensible Markup Language) schema. Therefore, the XSD Schemas can be used to send flight information to others in the form of XML encoded data.

2.2.2 AIXM

The Aeronautical Information Exchange Model is a specification designed to enable the encoding and distribution of the aeronautical information provided by the national Aeronautical Information Services (AIS), which was originally developed to meet the needs of the European AIS Database (EAD).

AIXM covers the requirements for the data necessary for the safety, regularity and efficiency of international air navigation. It has constructs for: aerodromes, navigation aids, terminal procedures, airspace and route structures, ATM and related services, air traffic restrictions and other data.

2.2.3 WXXM

WXXM stands for Weather Information Exchange Model, developed to pursuit the use of meteorological (MET) information in a data-centric environment. It is based on the Geography Markup Language (GML), an XML grammar used basically to express geographical features with vectors.

This Exchange Model also includes an XML representation of METAR data, the most common format for reporting observational weather information.
2.3 SWIM Registry

In line with the SWIM concept, the SWIM Registry aims at improving the visibility and accessibility of ATM information and services available through SWIM. This enables service providers, consumers, and regulatory authorities to share a common view on SWIM.

As depicted above, the registry enables the provider to publish information related to its services so that the consumer is able to discover them and obtain information required to fully use those services.

The registry is the source of reference, so it stores service descriptions, related standards, policies and certifications. It also contains a feature that enables the consumers to subscribe to services, receiving periodically related information and updates.

2.3.1 Current Services

The SWIM Registry currently supports about seventy different services. However, the ones that are worth mentioning are the officially services provided by Eurocontrol:

- Flight Preparation, aimed at easing the preparation phase of the flight plan, prior to its filing.
- Flight Filing, enabling the creation, update and cancellation of a flight plan.
- Flight Management, providing already filled flight plans.
- Measures Service, intended to provide services in the flow and capacity management domain.
- Airspace Structure, developed for querying and modifying airspace structure.
- Airspace Availability, including support of the Flexible Use of Airspace (FUA).

These services are all englobed and accessible via NM B2B Web Services, an interface provided by the Eurocontrol Network Manager (NM) for system-to-system access to its services and data, allowing NM customers to retrieve and use the NM information in their own systems.

On the other hand, there are some other interesting services like the FlightAware ADS-B data service. FlightAware is the largest flight tracking website in the world in term of users, offering real-time data through its ADS-B (Automatic Dependent Surveillance - Broadcast) data service.

## 2.4 Web Services

A Web service is a service offered by an electronic device (server) to another electronic device (client), communicating each other over computer networks like the Internet. Software applications written in various programming languages and running on various platforms can use web services to exchange data, without being tied to any operating system. For example, Java can talk with Pearl and Windows can talk with Unix applications.

Machine-to-machine interoperability is achieved with the use of open standards such as XML, SOAP, WSDL, UDDI and HTTP.

![Web services architecture](image)

XML (Extensible Markup Language) is used to encode all communications to a web service, so it’s the format of the data files. The way to validate if the XML data exchanged is following the correct format is describing the structure in a XSD (XML Schema Definition) file.
The protocol in charge of exchanging XML structured data is SOAP (Simple Object Access Protocol). SOAP relies on application layer protocols for message negotiation and transmission. In the case of web services, it relies on HTTP (Hypertext Transfer Protocol), the underlying protocol used by the World Wide Web.

It is also necessary to define rules for answering questions like: How one system can request data from another system? Which specific parameters are needed in the data request? What would be the structure of the data produced? All these rules for communication are defined in a file called WSDL (Web Services Description Language).

In the case that the client doesn’t know the server, and therefore does not hold a service’s WSDL, a web services discovery may be a suitable alternative. The UDDI (Universal Description, Discovery, and Integration) directory is an XML-based registry that defines which software system should be contacted for which type of data. It also provides access to the WSDL documents required to interact with the web services listed in its directory.

This kind of web services architecture has been around for a while and enjoys all of the benefits of long-term use. Nevertheless, it defines standards to be strictly followed and uses services interfaces to expose the business logic. For this reason, RESTful (REpresentational State Transfer) web services has become more popular. REST provides a lighter weight alternative and relies on a simple URL in many cases. Moreover, instead of using strictly XML permits other data format such as JSON (JavaScript Object Notation).
Chapter 3
Development

3.1 Testing ATM Web Services

It is always worthwhile to familiarize with existing services before designing our own. For this reason, the flight tracking service offered by FlightAware has been tested with temporally access credentials. Moreover, it has been tested in different ways to complete the overview of web services.

3.1.1 SOAP Requests with Python

The first test has been built in Python. Thanks to the SOAPpy library, Python is able to perform easily requests to existing web services, fully supporting dynamic interaction between clients and servers. Instead of reading WSDLs and generating SOAP queries manually, SOAPpy automatically parses the WSDL and populates the application with the available functions.

The sample code contains the FlightAware WSDL URL along with the credentials. It generates a SOAP action requesting the departures and arrivals of John F. Kennedy International Airport (ICAO Identifier ‘KJFK’) with the operation CountAirportOperationsResult. As it can be seen in figure 3.1, it returns a Python dictionary filled with the replied data.

```
<SOAPpy.Types.structType CountAirportOperationsResult at 139993138782360>: {'scheduled_arrivals': '1696', 'scheduled_departures': '1594', 'departed': '106', 'en route': '113'}
```

Figure 3.1: Python dictionary values for the SOAP request

3.1.2 REST and JSON

The alternative of the previous request is accessing using a light-weight REST inspired protocol that returns its responses encoded in JSON format. This allows the service to be used in environments in which it is inconvenient or impossible to invoke SOAP services, such as mobile phone applications, web browser applications, or server-side JavaScript environments.
To access any method, it is as simple as performing either a GET or POST request to http://flightxml.flightaware.com/json/FlightXML2/METHODNAME using standard CGI-style representation of the arguments. When requesting it must be supplied the credentials when asked as a "basic" Authorization HTTP header. For example, figure’s 3.2 URL is how the same previous request of John F. Kennedy airport (KJFK) in New York should be.

![REST access example using HTTP request](image)

**Figure 3.2: REST access example using HTTP request**

### 3.1.3 Jumpstart

Jumpstart is a System Wide Information Management (SWIM) demonstrator. It is an application written in C# for the Microsoft .Net framework in the form of a desktop-class application. It allows web services consumption for integration and testing purpose, being the de facto main demonstrator for SWIM both as a client example application and a data integrator. To sum up, Jumpstart is a tool to show all different groups of people what the possibilities are of SWIM and stimulate people/companies to start developing their own SWIM compatible tools.

It uses all different sources of data as input. Not only are available the sources previously mentioned as NM B2B Web Services from EUROCONTROL or FlightAware Surveillance data, but there’s also other types like information from meteo stations by Aviation Weather.

The test has been focused on monitoring the flights tracked via FlightAware’s ADS-B. Jumpstart is able to perform Web Service requests, obtain the corresponding JSON data, apply some information filters and print it using Bing Maps. figure 3.3 shows real-time aircraft flying in an specific area. Each color represents a different airline, specified with their ICAO code:

- Light Blue - Air Europa (AEA)
- Blue - Ryanair (RYR)
- Orange - Vueling (VLG)
- Red - Iberia (IBE)
- Yellow - Others

It includes as well other features such as filtering by aircraft id or setting the maximum time the point is remaining.
3.2 Infrastructure Design

Once gotten in touch with current web services, it’s time to build one. Starting with the design, it must be focused on the requirements and specifications listed at the beginning of the project.

In the first place, the project must consist of two modules:

- A web server containing the web service and a database
- A client able to perform requests to the server and Eurocontrol

The client has to request flight plans to Eurocontrol, this means accessing to NM B2B Web Services, in particular Flight Management Service. It also needs to be a java project to assure compatibility with MergeStrip application and hence provide a display to expose its results.

To simplify the client structure, it is desirable to build the web server based on Eurocontrol services. Since the server must store flight plans and deliver them, it should offer two services: Flight Filling Service and Flight Management Service.

Finally, it is necessary that both modules exchange standard ATM information. As the services share flight plans, the main data model used will be FIXM, specified in an XSD. As working with XSD and XML, the wiser choice is to follow with a SOAP and WSDL web service structure.
Figure 3.4 summarizes the design with all the specifications:

![Project infrastructure design schema](image)

**Figure 3.4: Project infrastructure design schema**

### 3.3 Web Service Deployment

In this section, the procedure followed in the main development of the project is described. The most detailed illustration is intended to be provided without code lines.

#### 3.3.1 Web Server

The first module of the development is creating the web service, which means building a web server from scratch. The deployment of the web service can be divided into three essential blocks: the creation of service logic, the connection with a database, and finally the publication of the server.

Reasonably, it is desired to work in the same environment in both client and server constructions. For this reason, the server will be built on Java with the help of an Eclipse IDE.
Creation of the service logic

Let’s remind that the more similar is this server to Eurocontrol’s the better, so that later the client will be able to communicate with both servers without changing the internal structure. To obtain Eurocontrol services specifications, the first thing to do is look for the WSDL in the SWIM Registry, containing the definition of them. This definition encloses the service endpoint, the operations available and the input and output types for each operation. In this case, the WSDL needed from Eurocontrol is called Flight Services, which holds the definition of three different services already exhibited: Flight Preparation Service, Flight Filing Service and Flight Management Service.

Sticking to the design, it is only required the filing and management services, more specifically, the operations to upload a flight plan (fileNewFlightPlan) and to retrieve them (queryFlightPlans, queryFlightsByKeys). Regarding the last ones, queryFlightPlans returns all the existing flight plans in the defined time window, while with queryFlightsByKeys it can be indicated the ID of the interested aircraft.

Nevertheless, owning the WSDL is not enough to start building this web service. The WSDL also requires an XSD that incorporates a more detailed definition for the input and outputs types of messages. This XSD is the one corresponding to the FIXM data model whose file name is FlightServices.xsd, version 19.0.0. However, this same file uses some other entries from AIXM and a shared basic file from these data models. Consequently, their XSD are necessary as well, AirspaceServices.xsd and CommonServices.xsd files. Because the data model comprehends a large extension of complex entries, in this document is not detailed the structure of the messages used, but they will be referred as Operation Request for the input and Operation Reply for their output.

Once got the modified WSDL and the corresponding XSD files, a web service can be created with specific tools. This is called building a ”Top Down” Web Service. The technology used in this project to achieve this is Axis2, from The Apache Software Foundation. Apache Axis2 is a Web Services / SOAP / WSDL engine built on Apache AXIOM, a new high performance, pull-based XML object model. Basically, its function is to create complete java classes from the entries of the WSDL and the XSD, including their relations and type restrictions. It creates as well classes to implement, containing the logic of the operations for each service. Particularly, two classes called FlightFilingServiceSkeleton and FlightManagementServiceSkeleton whose functions are fileNewFlightPlan and queryFlightPlans, queryFlightsByKeys respectively.

In a distributed computing environment, a skeleton acts as gateway for server side objects and all incoming clients requests are routed through it. The skeleton wraps server object functionality and exposes it to the clients. Also by adding the network logic ensures the reliable communication channel between clients and server.
Finally, a java project is created, FPServer, with a suitable folder structure and filled with the classes previously generated by Axis2. In this point, when starting to implement the skeletons functions, the need of a database quickly arises.

**Connection with the database**

A database will allow the uploaded flight plans to be stored and later retrieved. Since the server is being deployed locally, for the moment the database created is a simple MySQL DB, *FlightDB*, with a single table *FlightPlan*. Without details, this table contains the columns enough to store the important fields previously stipulated. The criteria followed to select these fields has been considering the ones defined as "cannot be null" in the FIXM XSD.

To reach this database from the server, the use of the JDBC is necessary. Java Database Connectivity (JDBC) is a Java-based data access technology that defines how a client (in this case the web server) may access a database, providing methods for querying and updating data. JDBC API mostly consists of interfaces which work independently of any database. This JDBC API is implemented by a database specific driver, for this project MySQL Connector/J, which is the official MySQL JDBC driver.

Taking advantage of the JDBC classes contained in the Java package *java.sql*, it is now possible to create connections and access the *FlightPlan* SQL table from the three functions *fileNewFlightPlan*, *queryFlightPlans* and *queryFlightsByKeys*. 

![Figure 3.5: Top down web service build](image)

![Figure 3.6: Java database connection](image)
Publication of the server

Finally, in this last block is achieved the visibility of the server built. The choice is relying on a Tomcat Server. Apache Tomcat is an open-source Java Servlet Container developed by the Apache Software Foundation (ASF) that provides a ”pure Java” HTTP web server environment in which Java code can run. In other words, it allows this FPServer java project to run and be accessed through the port 8080.

![Diagram of server and client](image)

Figure 3.7: Server accessibility and available services list

3.3.2 Client

The web service client built conforms the second module of this project development. Two main blocks are presented: its creation and the integration into the company’s platform.

**Generation of the client classes**

One more time Axis2 is the key for the creation of the java classes. With Apache Axis2 correctly configured on one hand and the WSDL on the other, the generation is instantaneously done using the wsdl2java terminal command. This time, though, no top-down service building option is selected.

The first part of the result are the same classes obtained in the generation of the server, the ones representing each XSD entry or FIXM type. Secondly, two other stub classes also appear, `FlightFilingServiceStub` and `FlightManagementServiceStub`. The client side object participating in a distributed object communication is known as a stub. The stub acts as a gateway for client side objects and all outgoing requests to server side objects. The stub wraps client object functionality and by adding the network logic ensures the reliable communication channel between client and server.
In conclusion, when a caller wants to perform remote call on the called object, it delegates requests to its stub which initiates communication with the remote skeleton.

All the classes generated by Axis2 are once again distributed into a new java project, whose name is FPClient. Lastly, a new class called Client is the one containing the logic and playing the caller role. It’s in charge of instantiating the stub, initializing the communication, managing the reply and returning the desired parameters.

![Figure 3.8: Distributed object communication](image)

### Integration into MergeStrip

Before moving forward, a brief introduction to Maven is essential. Apache Maven is a software tool used for building and managing any Java-based project. Making use of the concept of a Project Object Model (POM), Maven accomplishes a standard way to build projects, a clear definition of what the project consists of, an easy way to publish project information and a way to share JARs across several projects. Generally speaking, a project contains code lines, configuration files, dependencies and all of other little pieces that come into play to give code life. In the Maven world, a project need not contain any code at all, merely a pom.xml. In other words, POM is an XML representation of a Maven project held in a file.

Regarding MergeStrip, it’s a Maven project with its related POM file representation. Taking advantage of this fact, the integration process of the client is simplified. In the first place, the FPClient java project must be converted into a Maven project, which is basically create a pom.xml file and include all the JARs dependencies. To enable other projects to reach these project’s functionalities, it is necessary to add its files to the local Maven repository running `Maven install`. In this point, the only left to do is open the MergeStrip project and add a dependency line on its pom.xml interrelating the FPClient’s POM.

Finally, some modifications to MergeStrip’s logic should be included. Briefly, MergeStrip detects flying aircraft around the airport and displays them using different colors, blue for the ones arriving at the current airport and gray for the others. Before displaying an aircraft, an arrival or enroute flag is assigned to it. To resolve so, a `queryFlightsByKeys` request, specifying the aircraft id, is sent to the FPServer using the Client class of the FPClient project.
To sum up, in figure 3.9 it is depicted the whole performance of the service, integrating both server and client modules and offering a full and consistent view of its communication.

Figure 3.9: Service infrastructure result
Chapter 4

Results

This chapter encloses tests and results. In order to obtain an exhaustive evaluation of the developed service contrasting validations must be done, starting with a independent module and ending as a whole.

4.1 Server Validation with SoapUI

SoapUI is the world’s most widely-used open source API testing tool for SOAP and REST APIs. It offers SOAP Web Services functional testing, REST API functional testing, WSDL coverage, message assertion testing and test refactoring. It is the de facto method for ensuring quality when developing APIs and Web Services. Therefore, SoapUI enables the validation of the web service built, which is basically the FPServer java project running under a tomcat server on the port 8080.

The validation procedure is as simple as creating a new SOAP project and select the option from WSDL. When the web service WSDL is added, enable the TestSuite creation and continue. The SOAP project will be automatically built and a TestCase will be created for each service operation. TestCases act as plain clients, allowing to easily modify the XML content and send/receive SOAP operations through the service endpoints provided by the WSDL.

Although an explicit validation for each service and operation is done, figure 4.1 depicts only the results obtained with the fileNewFlightPlan operation, which is enough to grant a general picture. On the left it is shown the structured soap operation request containing the xml flight plan, while on the right it is displayed the soap response with a VALID.

4.2 Client Check using Eclipse

Once checked the performance of the web server with SoapUI, it follows an analysis of the client module. It was said in the development chapter that the FPClient java project’s logic relies in the class Client, acting as a caller through its functions. Nevertheless, an static main method is needed to allow the project to run. Such method consist of a Client object initialization specifying the endpoint, a call to its functions with the required parameters and a final print of the obtained response objects.
Despite showing in figure 4.2 only the result of `queryFlightsByKeys` operation, it must be alleged that the same validation has been done for the other two operations remaining. The figure portrays the Client class at the top and the printed response under it, which corresponds to the summarized flight plan related to the requested aircraft id.
4.3 Interoperability Test with Python

When describing web services an interesting feature appeared. Enabling machine-to-machine interoperability without being tied to any language or operative system was promised on the service. Consequently, it has been decided to test this quality building a Python client.

The Python client has been developed using a pure-python module named Zeep. Zeep inspects the WSDL document and generates the corresponding code to use the services and types in the document. This provides an easy to use programmatic interface to a SOAP server and also offers support for HTTP Get and Post bindings. Parsing the XML documents is done by using the lxml library, which is the most performant and compliant Python XML library currently available. This results in major speed benefits when processing large SOAP responses.

The code used to perform the test consists of a Zeep client creation detailing the endpoint, an operation call through the client and finally a JSON serialization of the response. This JSON data is printed and also saved as an external json file.

Figure 4.3 illustrates the returned summarized flight plans as a result of a queryFlightPlans request, even though as before other operations has been tested too.

Figure 4.3: Python client queryFlightPlans response on terminal
4.4 Web Service Evaluation from MergeStrip

In this last section is evaluated the final performance of the whole system. Not only because it includes the integration into MergeStrip, but also it’s the first test where interacts two physical machines. Although the server still resides in the same development computer along with its database, in this case the caller is running in another device inside MergeStrip. Both PCs are contained in the same LAN, hence they easily reach each other. Not forgetting the server is running on the port 8080, the only modification needed is replacing the endpoint of the Client constructor from localhost (127.0.0.1) to the current IP on the net.

Running MergeStrip from Eclipse automatically starts calling the server for flight plans of the aircraft on its list, one by one. In this specific test only four flight plans were created to be stored on the server with the appropriate aircraft identification and destination airport. Figure 4.4 shows the four flight plans received and printed by the Eclipse console, while in figure 4.5 they are displayed in blue around Budapest Ferenc Liszt International Airport.

MergeStrip’s view consists of three main blocks. The first one on the top left is a spatial representation of the aircraft on a map. On the bottom there are two dynamic timelines, one contains aircraft from 0 to 30 minutes to the destination point while the other has an alternative unit measure with an smaller scale. Finally, the top right represents the relation between altitude and distance to the reference point.

![Figure 4.4: Eclipse console with the flight plans received](image-url)
Figure 4.5: MergreStrip’s display of the arrivals

It must be remarked that this test has been done under a MergeStrip simulation. Consequently, instead of detecting the aircraft they are being generated with an specific algorithm.
Chapter 5

Budget

This project involves research, design, development and test, but not an operational implementation. This kind of implementation hasn’t been possible since it relies on other company’s projects on development (UTM system, MergeStrip) as well as on external authorizations (NM B2B Web Service certificate, FlightAware credentials). Thus, physical servers or any electronic devices weren’t taken into account in this budget.

Regarding this mentioned authorizations, the Eurocontrol NM B2B Web Services’ certificate agreement in process is provided free of charge. On the other hand, FlightAware credentials used for the initial tests were borrowed as a free month trial.

To sum up, in this budget it has only been included the salaries, both student’s and supervisor’s. Considering the salary of a junior engineer as 15 €/h and the senior’s as 40 €/h, calculated with 400 working hours, of which the supervisor is supposed to be involved 1 hour a week:

- Student: 15 €/h x 400 h → 6.000 €
- Supervisor: 40 €/h x 20 h → 800 €

Total: 6.800 €
Chapter 6

Conclusions and Future Work

In this thesis a new concept of ATM information management has been presented, the SWIM program. It has also introduced the reference, data models associated and the technology beside it. From all this, an infrastructure design has been proposed to satisfy some of the company’s projects needs. Such design has later been developed, integrated into an application and successfully tested. In conclusion, this project was able to show the capabilities of the SWIM concept and has built a bedrock for the company to start developing its own SWIM compatible tool.

During the whole process of the project a lot of obstacles constantly arose. These obstacles usually required the adaptation to a new scenario and some decision-making to stick to the project requirements. For example, the project deadline interfered with the final validation of the client built. It was created focused on performing equally when communicating with the local server and Eurocontrol’s. This feature couldn’t be checked due to the NM B2B Web Services certification request procedure and paperwork still in process.

Regarding more technical difficulties that appeared, working based with the existing Eurocontrol services and particularly managing the FIXM data model are the ones worth mentioning. The information exchanged inside the request and response messages can be so large and complicated that a huge class library is needed inside the project. Although the information desired by the company’s application was minuscule, some other non-interesting fields were also required to ensure the communication. In other words, FIXM contains some fields in every object that cannot be null, which forced the management of great amounts of structured data. In addition, the data model and so the .xsd files couldn’t be modified to ignore the undesired information entries because, as reminded previously, the client must perform correctly as well when accessing to Eurocontrol services.

To sum up, the future possible improvements can be divided in two lines of development. On one hand, follow the UTM system development and publish the web service along with the database into a real server. On the other, wait for Eurocontrol certification and validate the client in MergeStrip. In both cases, it might be needed to modify the web service by supporting other services or adding new operations to the existing ones.
Bibliography


[16] NextGEN. *System Wide Information Management (SWIM)*. Product portfolio, Federal Aviation Administration, 2016.


Acronyms

ADS-B Automatic Dependent Surveillance - Broadcast
AIRM ATM Information Reference Model
AIS Aeronautical Information Services
AIXM Aeronautical Information Exchange Model
ANSP Air Navigation Service Provider
API Application Programming Interface
ASF Apache Software Foundation
ATM Air Traffic Management
AXIOM Axis Object Model
B2B Business to Business
DB Database
EAD European AIS Database
FAA Federal Aviation Administration
FIXM Flight Information Exchange Model
FP Flight Plan
FUA Flexible Use of Airspace
GML Geography Markup Language
HTML Hypertext Transfer Protocol
ICAO International Civil Aviation Organization
IDE Integrated Development Environment
JAR Java Archive
JDBC Java Database Connectivity
JSON JavaScript Object Notation
LAN  Local Area Network

MET  Meteorological

METAR  Meteorological Aerodrome Report

NM  Network Manager

POM  Project Object Model

REST  Representational State Transfer

SESAR  Single European Sky ATM Research

SOAP  Simple Object Access Protocol

SQL  Structured Query Language

SWIM  System-Wide Information Management

UAS  Unmanned Aircraft Systems

UDDI  Universal Description Discovery and Integration

UML  Unified Modeling Language

URL  Uniform Resource Locator

UTM  UAS Traffic Management

WSDL  Web Service Description Language

WXXM  Weather Information Exchange Model

XML  Extensible Markup Language

XSD  XML Schema Definition