



UNIVERSITÀ
DEGLI STUDI
FIRENZE

DOTTORATO DI RICERCA IN
INFORMATICA, SISTEMI E TELECOMUNICAZIONI

TELEMATICA E SOCIETÀ DELL'INFORMAZIONE

CICLO XXVIII

COORDINATORE Prof. Nesi Paolo

*Automatic Train Operation System for light rail and metro using
a model-driven approach and Social Network Monitoring to
quantify public attention on services and events*

Settore Scientifico Disciplinare ING-INF/05

Dottorando

Dott. Menabeni Simone

(firma)

Tutori

Prof. Nesi Paolo

(firma)

Prof. Fantechi Alessandro

(firma)

Coordinatore

Prof. Chisci Luigi

(firma)

Anni 2012/2015

Table of contents

Preface	6
Part 1.....	8
1 Introduction to Railway Signalling Systems.....	8
1.1 Railway Signalling Systems technology.....	8
1.1.1 ERTMS/ETCS standard	11
1.1.2 Braking Model and Speed Profiles	15
1.2 CBTC Overview	16
1.2.1 Domain Analysis and existing technologies	17
1.2.2 Reference Standards	18
2 ATO System Analysis.....	21
2.1 Main challenges	21
2.2 ATO Preliminary System Specification.....	22
2.3 ATO System Requirements Specification.....	26
2.3.1 ATO-ATS Communication Protocol.....	30
2.3.2 ATO-TRAIN Communication Protocol.....	30
2.3.3 ATO-ATC Communication Protocol	30
2.4 Initialization Phase	31
2.4.1 Procedure Start of Mission	33
3 ATO System Modelling	43
3.1 Model Based Systems Engineering.....	43
3.2 IBM Rational Rhapsody.....	44
3.3 Package Init Design	46
4 Model Verification and Testing	55
4.1 Adopted Methodology.....	56
4.2 Critical aspects of the model	57
4.3 Model Testing	58
4.4 Integration Testing.....	59
Part 2.....	60

1	Introduction to Social Media Monitoring.....	60
1.1	Social Media Monitoring.....	60
1.2	Twitter.....	65
2	Twitter Vigilance Analysis.....	67
2.1	Requirements.....	68
2.2	Twitter API	69
2.3	Main challenges	82
3	Twitter Vigilance Design	84
3.1	Backend.....	84
3.1.1	Requirements	85
3.1.2	Backend Architecture	86
3.1.3	DB structure.....	86
3.1.4	Crawler.....	98
3.1.5	Scheduler	100
3.1.6	Other Scripts.....	104
3.2	Dashboard.....	105
3.2.1	Requirements	105
3.2.2	Frontend Architecture	107
4	Testing and Validation	123
	Conclusions.....	127
	Bibliography.....	129
	Table of Figures	133
	List of Tables	135

Preface

The research activities during the entire course of PhD was carried out at the DISIT laboratory (Distributed Data Intelligence and Technologies) of DINFO (Department of Information Engineering) of the University of Florence. The research activity is essentially divided into 2 parts and concerns two arguments:

- Analysis and Modelling of Automatic Train Operation Systems
- Social Network Monitoring

The first part of the research was conducted within the project TRACE-IT in collaboration with ECM, with the Department of Industrial Engineering of the University of Florence and the Institute of Science and Technology of the CNR Pisa. Inside the lab DISIT, the activity was carried out together with other two PhD students, Giacomo Martelli and Mariano DiClaudio, responsible respectively for designing communication protocols and the management system for train running.

The research project deals with the study, design, development and testing of systems of protection of the train running (Automatic Train Protection, or ATC Automatic Train Control, ATC) of railway vehicles that use leading edge technologies and are applicable both in the context of interoperable European Rail System and in applications of light rail and metro. In particular, the research has focused on protection systems and automatic drive of train in the field of light rail and subway called CBTC (Communication Based Train Control).

Starting from the study of standards and products already on the market carried out during the first year of the course has decided to model and implement an ATO (Automatic Train Operation) able to be at the forefront over existing systems. To achieve this has decided to create a system compatible with an ATC system compliant to ERTMS / ETCS standard. In particular, the ATO system is designed to interface with the ERTMS / ETCS Level 2: to meet this requirement, the system must implement both the behaviour of the driver interacting with the DMI (Driver Machine Interface) either the behaviour of the DMI itself. In addition to these requirements the ATO must comply with other specifications such as the initialization of the entire system on board the train, the automatic adjustment of the running of the train, the opening and closing of the doors, etc.

My research has focused on the analysis, modelling, implementation, and testing of the initialization phase of the onboard system.

In the second part of the research activity, has designed and built a platform for scanning Social Network: for the purpose are used the Twitter API to create a database of messages. This platform is developed by DISIT Lab of the University of Florence and allows to monitor Twitter channels and slow and explosive events on Twitter. The simplest channels include the tweets corresponding to a single user, to a single hashtag, or a single keyword. Channels more complex provide dozens of complex queries and combined between keywords, users, hashtags, etc. For the events the platform monitors tweets per day / hour or per day / week depending on whether the event is explosive or slow. So we can compute metrics on tweets and retweets relating to a certain event or channel. In addition to the main process for the recovery of the messages will be realized a number of secondary processes useful to the realization of a dashboard for displaying the results of the scan and for the analysis of the messages retrieved. This project is the result of a collaboration between UNIFI DISIT laboratory, CNR IBIMET and LAMMA to investigate and build specific metrics and a dashboard reliable to monitor the tweets that refer to the weather. The study has helped to develop the Twitter Vigilance platform.

The analysis of social networks has become the main source of news, comments, opinions on any subject. The analysis shows that there are already several instruments, paid or not, which analyze and extract information from various social channels. Twitter Vigilance was designed as a platform for the search of messages on Twitter and for the analysis of such messages both from a numerical point of view (to highlight the daily peaks), both from a semantic point of view (to identify what refer spikes of tweets).

Twitter Monitoring through specific channels offers the opportunity to gain some understanding of the perception of the people on environmental issues and on other issues. Twitter Monitoring also has useful ideas to government officials on reliable sources of information shared via Twitter and tips on how to improve communication to citizens on these issues.

The thesis is divided into 2 parts which reflect the topics of the research activities carried out within the PhD. The first part concerns the analysis, modeling and validation of the initialization phase of an ATO system for light rail and metro and is divided into four chapters: Chapter 1 is the introduction and includes state of the art on railway signaling, Chapter 2 describes the analysis of the ATO system that identifies the main issues of the project and in particular of the initialization of the onboard system, Chapter 3 describes the modelling activity of the initialization phase and Chapter 4 presents the validation and testing of the designed system. The second part concerns the analysis, design and validation of a platform for Social Network Monitoring and is divided into four chapters: Chapter 1 is the introduction and includes state of the art the social network monitoring with a focus on Twitter, Chapter 2 describes the analysis of the problem by identifying the main critical issues due mainly to the Twitter API, Chapter 3 describes the design of the Twitter Vigilance platform (architecture, database structure and statechart of individual processes) and Chapter 4 presents the validation of the developed system.

Part 1

1 Introduction to Railway Signalling Systems

1.1 Railway Signalling Systems technology

Signalling is one of the most important parts of the railway system. The signalling adoption derives from the need to ensure safety in train movement and in railway traffic control in order to prevent trains from colliding. Over the years the signalling technology had a considerable development starting from hand signals to the most modern systems for trains separation [1].

In signalling railway a key concept not changed over the years exists: *Trains cannot collide with each other if they are not permitted to occupy the same section of track at the same time* [2]. On this concept all railway safety systems have been implemented, including the most advanced systems nowadays used.

From 1850 the railway lines have been divided into sections called block sections (or blocks) and only one train was permitted in each block at a time (in most cases this rule is still valid nowadays). At the beginning, men at specific intervals along the rail lines with a stopwatch signalled that a train was going to pass with hand signals.

By 1900 mechanical semaphores were placed along the rail lines in order to help the staff. With the invention of the telegraph and then of the telephone, the staff could send a message (first a certain number of rings on a bell, then a telephone call) to confirm that a train had passed and that a specific block was finally clear. About in 1930 the first optical signals were introduced and the whole system was called phone block system. From the 1930s, the semiautomatic block was introduced with hand signals replaced by fixed mechanical signals. In this type of signaling one block (or protected section) usually was adopted along the railway line between neighboring stations. So only one train could occupy the section at a given time and the departure of a train from one station was possible only when the section was free.

Both the train departure and arrival required the assistance of duty officers: at depart station the officer received special blocking signal from the neighbour station duty offices

while at arrival station the officer ensured that the line was free and pressed a special button to send arrival signal to the neighbour station [3].

Nowadays, railway signalling is based on automatic blocks where manual intervention is not required. The railway line is still composed of a series of block sections of a certain length and, depending on typology of blocks, we can have fixed-block and moving-block signaling systems. These systems are normally made up of two parts, a wayside equipment and an onboard equipment. The main task of the wayside equipment consists in communicate to the train (to onboard equipment) the distance that can be covered in safety. This distance is generally called *Movement Authority (MA)* and is used by the onboard equipment to compute the speed curve (normally called *braking curve*). This curve must be respected by the train in order to be able to stop before reaching the end of the MA [4].

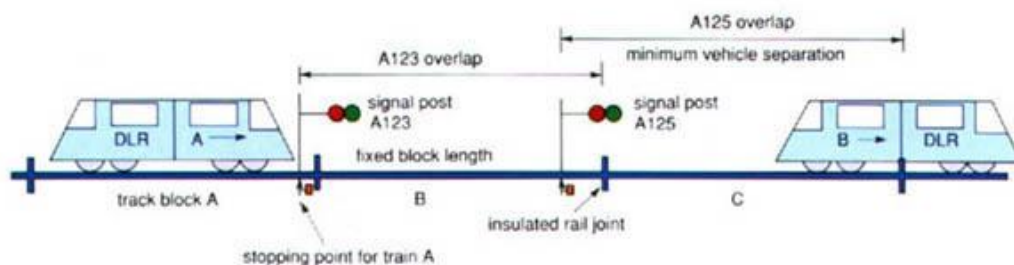


Figure 1: Fixed-block signaling system

In a *fixed-block system* (Figure 1) the length of block sections is fixed and is established when the system is deployed. So in these systems each block length defines the Movement Authority that is therefore fixed. This information is received by the onboard equipment at the beginning of each block in order to compute the braking curve according to this distance. The train will operate the emergency brake before reaching the beginning of the next block if it is occupied by another train.

Two equipments are primarily used to detect the presence or transit of the train in a particular section: track circuit and axle counter or a combination of two.

Track circuits are electrical circuits separated by isolating joints made of electromagnetic devices also used to inform signallers and to control relevant signals. The transit of a train on the track triggers the electrical contact between the two rails, so the circuit is closed. Consequently a relay is characterized by zero current and the block signal is set at danger or occupied. In railway systems one of the major priorities is the railway safety, so fail-safe operation is crucial. In this case the track circuit is designed to indicate the presence of a train when failures occur. Nevertheless reliability and quality of railway services are

essential, so false occupancy readings are disruptive to railway operations and are to be minimized.

The jointless track circuits are a track circuit typology and are obtained turning to alternating current circuits. In this case the blocks can be separated by an inductive shunt connected across the rails, avoiding the need for insulated joints. In non-electrified area alternating current track circuits use different frequencies, combinations of frequencies or modulated frequencies.

The Axle Counters are equipments placed at the start and the end of each block section. This equipment uses two electronic wheel sensor systems to detect all the axles of rolling stock travelling on a track and their direction of travel, using two electronic wheel sensor systems. The result for the axles counted in are then compared with the result for those counted out supplying the status of the track section (if it is free or occupied).

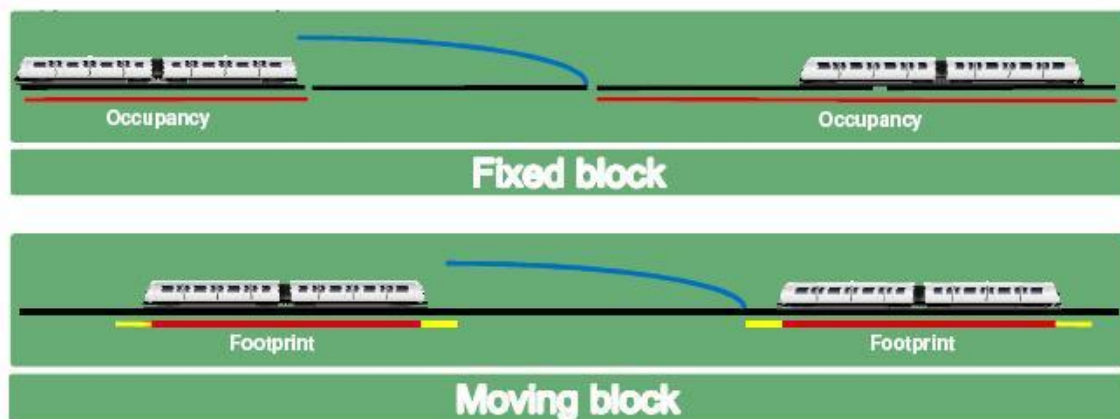


Figure 2: Safety distance between trains in fixed block and moving block signalling systems

To reduce the minimum headway between consecutive trains in order to further improve the line capacity the Moving block system has been conceived, where each block section has a length not established a priori. Unlike fixed-block systems, the train receive a prolonged MA up to the rear of the vehicle in front (Figure 2) or any obstacle that prevented the MA from being set to its maximum length. So the clear (safe) distance ahead of each train must be calculated continuously and the appropriate speed, braking, or acceleration rate must be sent to each train. This requires continuous two-way communication with each train, and precise knowledge of a train's location, speed, and length, and of fixed details of the line—curves, grades, interlockings, and stations. For this reason moving-block signaling systems are also called transmission-based or communication-based signaling systems.

Without track circuits to determine block occupancy, a moving-block signaling system must employ an independent technique to accurately locate the position of the front of a train,

and then use look-up tables to calculate its end position from the length associated with that particular train's identification. The technique used is based on the position information capture through active and passive markers (transponders) along the tracks, and train-borne tachometers and speedometers. These are interrogated by a radio signal from each train and return a discrete location code. Satellite-based systems cannot be used because they will not work in tunnels. In such a system it is therefore necessary to use computer located on each train, at a central control office, dispersed along the wayside, or a combination of these. The most common arrangement is a combination of on-board and central control office locations.

Nowadays a moving-block signaling system includes a train control systems, or more properly Automatic Train Control (ATC) to increase the railway safety. Automatic train control usually encompasses three subsystems: Automatic Train Protection (ATP), Automatic Train Operation (ATO) and Automatic Train Supervision (ATS). Automatic Train Protection is the basic separation of trains and ensures that the MA is always respected by train. The respect of the MA implies, also, that the speed limit is not exceeded by the train. To guarantee this, the ATC computes a dynamic braking curve based on train data such as train position and speed. The ATO adds speed control and often automatic train management. The automatic train operation functionalities are various, such as: regulation of train speed within limits imposed by the ATC sub system ensuring passenger comfort as established by operating policy, energy consumption management and optimization, automated coupling & decoupling, train movement control with regard to speed, acceleration, deceleration, and jerk and others. These features allow the automatic control of train movement without drivers ensured by the onboard Automatic Train Operation (ATO) system in combination with ATC. The supervision and management of train traffic is provided by Automatic Train Supervision. ATS can integrate also other functionalities, including train depot management, train wakeup/sleep, integrated maintenance, incident report/replay and train routing. Automatic train protection and automatic train control maintain the fail-safe principles of signaling and are referred to as vital or safety critical systems. Automatic train supervision cannot override the safety features of these two systems, and so it is not a vital system.

The ATC is combined with Interlocking systems (IXL) which are a set of signal apparatus placed on the track in order to ensure a safe journey and avoid all risks of conflict between train paths. This is performed through an arrangement of track devices such as junctions , derails and crossings.

1.1.1 ***ERTMS/ETCS standard***

Nowdays with ATC/ATC denomination we refer to all the automated systems that protect the driver and then the train from possible overspeed or exceed of stop signals. Over the years the various European countries have realized different ATP/ATC systems that have

been developed according to the own national requirements and technical standards, and operated according to the own operating rules.

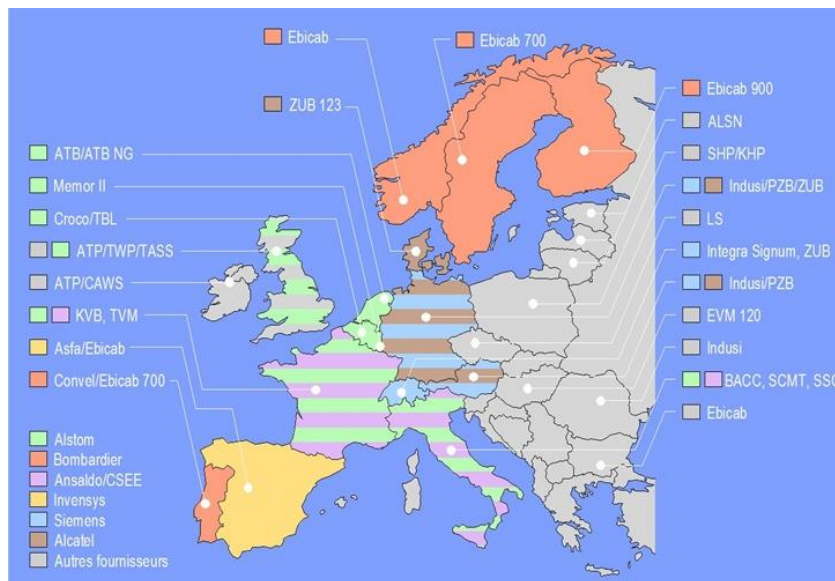


Figure 3: European ATC/ATC systems before ERTMS/ETCS

This has led to the spreading of incompatible train protection and control systems impeding the cross-border operation of European railway traffic. There were also other technical differences among the European countries relating to rail gauge, electricity voltage, rolling stock design, etc. For example, in Italy the BACC system based on conventional coded track circuits employed at two different carrier frequencies, was used. The goal was to manage two train classes, those operating either above or below 180 km/h [5].

Establishing common rules for the free movement of train in all European countries became essential with the advent of European integration and with this remarkable diversity of European ATC systems.

Following the decision taken by the European Transport minister in December 1989, the EU embarked upon a project to analyze the problems relating to signalling and train control [6][7]. At the end of 1990, the ERRI (European Institute of Railway Research) began to think to develop a common interoperable ATC/ATC system, which could be adopted in all European countries. Implementing a common interoperable platform for railways and signalling systems to ensure the interoperability of the European rail network's was the final goal of the International standard programme ERTMS. The main objectives of interoperability are based on the need to simplify, improve and develop international railway transport services, contribute towards gradually creating an open and competitive domestic market for the supply of railway systems and construction, renewal, restructuring and operative services, and establish standardised European procedures for assessing

conformity with interoperability requirements [1]. To define the TSI (Technical Specification for Interoperability) a group of railway experts called ERTMS Group was created, consisting originally of DB, FS and SNCF, but later joined by other railway European companies. To finalize the TSI in the summer of 1998 the UNISIG union was formed, comprising the European Signaling companies Alcatel, Alstom, Ansaldo Signal, Bombardier, Invensys Rail and Siemens. The standardized, interoperable command and control ATC/ATC system was called ERTMS/ETCS, or simply ETCS. In Europe this command-control and signalling system was gradually introduced under the ERTMS/ETCS (or simply ETCS) denomination. Thanks to this system, from the beginning of the 21s century interoperability of the European rail network is going to be guaranteed.

ETCS is divided into different functional levels as established by the set of specifications Baseline 3 (SRS 3.3.0) [8]. The definition of the levels depends on how the railroad is equipped and on the way through information is transmitted to the train. Furthermore a train fitted with complete ERTMS/ETCS equipment and functionality can operate on any ETCS route without any technical restrictions.

The four levels identified by standard are the following:

- Level 0 = level corresponding to an ETCS vehicle running on a non-ETCS route. In this case the trackside signals must be still observed and the trainborne equipment has the only task of monitoring the train speed.
- Level 1 = the ERTMS/ETCS system is superimposed to the traditional signalling equipment. In this case the traditional trackside occupancy controlling devices (track circuits) detect the train position and are linked with the interlocking. Eurobalises are usually used to transmit from track to trains fixed or variable data as the permission to cross one or more block sections (Movement Authority). With these data the on-board equipment continuously monitors and calculates the maximum speed and the braking curve. Optionally, Euroloops or radio infill units may be used.

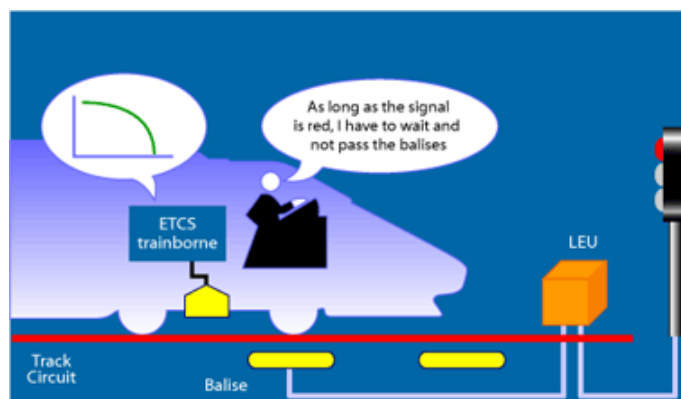


Figure 4: Level 1

- Level 2 = the ERTMS/ETCS system uses a GSM-R radio channel to allow a continuous exchange of data between the trackside equipment (Radio Block Centre (RBC)) and the trains. Train movements are monitored continually by the RBC that generates the correct movement authorities for the different trains in the section. MA are then transmitted to the trains together with speed information and route data. Most of the signals are displayed in the trainborne cab making the lineside signals no longer strictly necessary. The on-board equipment continuously monitors the transferred data and the maximum allowed speed. The trains report their position to the Radio Block Centre via the GSM-R communication channel and use the Eurobalises as passive positioning beacons or electronic milestones. Between two positioning beacons the train determines its position via sensors. The positioning beacons are used in this case as reference points for correcting distance measurement errors. For the train integrity, supervision the traditional signalling devices (track circuits) are still kept.

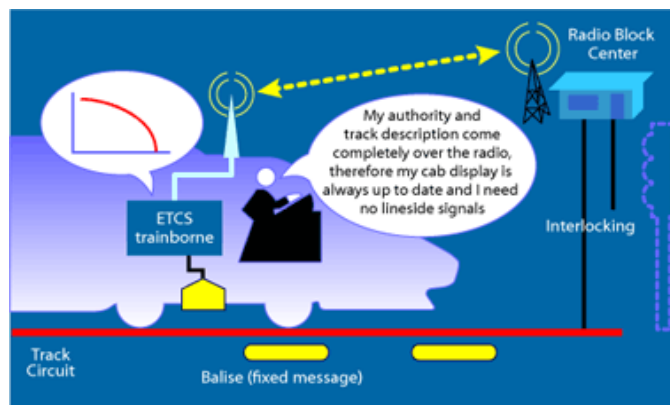


Figure 5: Level 2

- Level 3 = ETCS Level 3 provides an implementation of full radio-based train spacing. So the line-side signals as well as the trackside occupancy checking devices are no longer required. The location of trains is determined by the train-side odometry and reported to the trackside Radio Block Centre via GSM-R radio transmission. The movement authority is then given on the information relating to the position of the train, based on the actual distance of a train from the next. Nevertheless this level is currently under development.

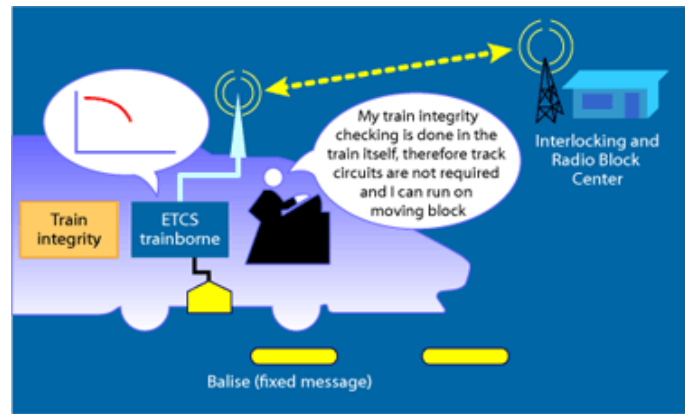


Figure 6: Level 3

The continuous exchange of data between an ERTMS L2 fitted train and the trackside sub system is ensured by the GSM-R mobile communications. This mobile radio system, that is used exclusively in the railway sector, allows a constant contact between the crew and the ground (service communications and emergency management) for the exchange of data between trackside and on-board systems, by establishing a circuit-switched connection. Other communications standard have been considered by the UNISIG union. For example, the wireless and connection-oriented approach is often considered as the bottleneck of the signalling system, which considerably limits the possible number of voice and data connections in each cell of the network at the same time and it can cause a deadlock of the system, if the number of users will rapidly increase (e.g. accidents, freight depots, lines with a high and dynamic volume of traffic).

Today Spain (3800km), France (2000km), Germany (1600km) and Italy (1000km) are currently the most equipped states. In parallel to European adoption, other countries (China, Taiwan, South Korea, India, Algeria, Libya, Saudi Arabia, Mexico, New Zealand or Australia) have launched major ERTMS investment programs.

1.1.2 **Braking Model and Speed Profiles**

The ERTMS/ETCS train control system includes an Automatic Train Protection (ATP) subsystem based on the concept of braking model. The latter is a mathematical model that, knowing the current speed of a vehicle, the target distance and the characteristics of the braking system, is able to predict the trend of the speed related to space. So once this model is known it is possible to determine instant by instant the maximum speed at which the train can travel so that it can stop safely before the dangerous point, that may be represented by a stop signal. The maximum speed of the train must take into account two types of constraints that define two specific curves. The first one called *ServiceSpeedProfile* considers the sequence of stations where the train will arrive respecting the fixed time. The second

one called *SignalingSpeedProfile* instead refers to the presence of curves, turnouts and possible temporary interruptions along the railway line.

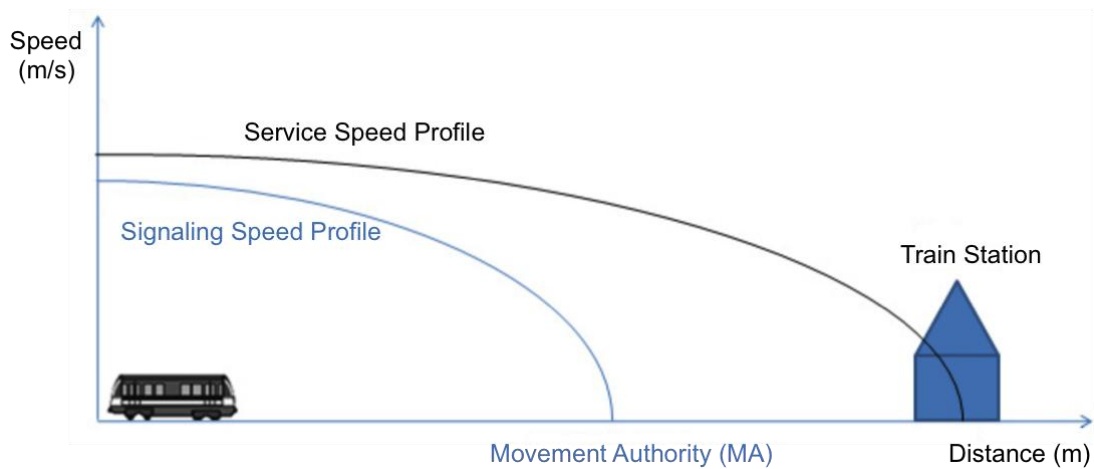


Figure 7: Service Speed Profile and Signaling Speed Profile

Figure 7 shows an example of the two curves just described, that regulate the performance of the train. It is possible to observe how in this case the MA (Movement Authority) does not match with the target distance, and then the limit to which the train will have to refer is that defined by the signaling speed profile. In fact in order to ensure safe conditions the speed of the train will always be less than that defined by the most restrictive of the two profiles.

To ensure that the onboard system can calculate the allowed speed, it must receive from the trackside equipment the following information:

- Movement Authority (MA), the space that the train is authorized to travel in safe condition.
- Speed limit profiles, namely the limitations of the signaling static and/or dynamic.

At the same time the trackside system has to know by the onboard equipment the following information:

- The current position of the train along the line and the travel direction.
- The status of the railway line downstream of the location of the train.

1.2 CBTC Overview

The Communications-Based Train Control (CBTC) systems are novel signalling and control systems in urban context for light rail (e.g., tramway), heavy rail (e.g., metro) and APM

(Automated People Mover, e.g., Airport metros)[9] [10]. These systems give operators precise control in the movement of their trains, allowing more trains to run on the line at higher frequencies and speeds in total safety — with or without drivers. The goal is to improve capacity, efficiency, reliability, safety and operational flexibility of the train, and to reduce operating costs.

1.2.1 *Domain Analysis and existing technologies*

The traditional systems that do not use a CBTC approach are normally referred as fixed block systems, since the railway is divided into sections of track, which are separated by signals [2]. A train is not allowed to enter a given track section (block) before the preceding train has cleared it. The presence of trains is detected by the track circuits. Therefore, the position of the train is based on the accuracy of the track circuit, and the information provided to the train is limited to the one provided by the wayside signals.

The CBTC systems, also referred as moving block systems, don't require the use of track circuits because the train position is provided by the onboard equipment with a high precision. Furthermore, much more control and status information can be provided to the train exploiting a continuous wayside-to-train and train-to-wayside data communication. In this way the CBTC is aware at any time about the exact train position and speed. Currently, most of CBTC systems implements this communication using radio transmission.

Thanks to the moving block principle, CBTC is able to reduce the distance between two trains running in the same direction (this distance is normally called headway). In this case the minimum distance between successive trains is no longer calculated based on fixed sections, as occurs in presence of track circuits, but according to the rear of the preceding train with the addition of a safety distance as a margin. This distance is the limit distance (Movement Authority (MA)) that cannot be overcome by a running train. In this way a safe operation of trains is ensured.

The CBTC uses the onboard Automatic Train Control (ATC) system to ensure that the MA is always respected by train, in addition to ensure a continuous protection of the train in every aspect. The respect of the MA implies, also, that the speed limit is not exceeded by the train. To guarantee this, the ATC computes a dynamic braking curve based on train data such as train position and speed.

The automatic control of train movement without drivers is ensured by the onboard Automatic Train Operation (ATO) system in combination with ATC. This automatic control allows to lead the train from one station (or predetermined operational stopping point) to the next, respecting the required speeds of the track and the operating conditions ensuring safe operation [11]. Other ATO functionalities:

- control train movement with regard to speed, acceleration, deceleration, and jerk;

- regulation of train speed within limits imposed by the ATC subsystem, ensuring passenger comfort as established by operating policy;
- management of programmed stops;
- management of Platform/Train doors;
- skip programmed stops;
- management of overshoot and undershoot;
- automated coupling & decoupling;
- energy consumption management and optimization.

From the architectural point of view the CBTC system is constituted by an onboard equipment and a wayside equipment. The first is installed on the train and forms the onboard subsystem (BSS). The second equipment is located at a station or along the line and forms the trackside subsystem (TSS).





1.2.2 *Reference Standards*

Currently, the different CBTC products are governed by IEEE 1474.1-2004 and IEC 62290 [12][13]. These implementations are offered on the market by different vendors such as Bombardier, Alstom, Thales, Invensys Rail Group, Ansaldo STS, and Siemens.

The IEEE 1474.1-2004 has been defined by the Communications-based Train Control Working Group of Institute of Electrical and Electronic Engineers (IEEE) and approved in 2004. Such standard defines the functional and performance requirements that a CBTC system shall implement. The functional requirements describe the functionalities of components of CBTC system identified in Automatic Train Protection (ATP), Automatic Train Operation (ATO) and Automatic Train Supervision (ATS). The ATO and ATS functions are considered optional by the standard. The performance requirements describe how well the functionalities are to be executed or achieved, or how well they are to be accomplished. In addition to these requirements, this standard also establishes the headway criteria, the system safety criteria and the system availability criteria applicable to different transit applications, including the Automated People Movers (APM).

The IEC 62290 is a standard defined by the International Electrotechnical Commission (IEC) gone into effect in 2007. This standard includes the fundamental concepts, the general requirements and a description of the functional requirements that the command and control systems in the field of urban guided transport, like the CBTC, shall possess. For these systems the standard establishes four levels of automation called Grades of Automation (GoA1 to Go4). The definition of Grades of Automation arises from apportioning responsibility for given basic functions of train operation between operations staff and system. For example, a GoA1 system simply enforces brakes when the driver violates the

dynamic braking curve. A GoA4 system ensures fully unattended operations without operations staff.

Grade of Automation	Type of train operation	Setting train in motion	Stopping train	Door closure	Operation in event of Disruption
GoA 1 	ATP with driver	Driver	Driver	Driver	Driver
GoA 2 	ATP and ATO with driver	Automatic	Automatic	Driver	Driver
GoA 3 	Driverless	Automatic	Automatic	Train attendant	Train attendant
GoA 4 	UTO	Automatic	Automatic	Automatic	Automatic

ATP - Automatic Train Protection ATO - Automatic Train Operation

Figure 8: GoA Levels

CBTC technology includes complex hardware and software systems that interact tightly with human users and are constrained by a number of national/international regulations and standards. Therefore the CBTC are critical systems with regard to the safety and reliability since a failure can lead to catastrophic consequences ranging from loss of profit to human lives. For this reason these systems must be certified according to the strict international guidelines and, in the field of railway signalling software systems, these guidelines are represented for software by CENELEC EN50128.

The CENELEC EN 50128 Standard, “Railway Applications: Software for Railway Control and Protection Systems” is part of a group of related Standards: the EN 50129 and the EN 50126 [14][15][16]. The latter ensures that the embedded software is suitable for use in safety-critical settings, while the EN 50129 provides guidelines for the electronics systems used for signaling. The EN 50126 and the EN 50129 are focused on the safety functions allocated to software, whereas the EN 50128 specifies the methods which need to be used in order to develop software which meets the demands for safety. The key element of this latter norm is the concept of software safety integrity levels (SILs). SIL defines the robustness degree in terms of protection level that the system must have against the failures both random type that systematic type. There are four safety integrity levels (from SIL1 to SIL4): S4 is associated with the highest level of integrity, while S1 is associated with the lowest. The SIL0 value is used for indicate that there are NO safety requirements.

The light rail metro systems are very similar to the conventional rail, but with some important differences:

- are closed systems;

- are constituted by a geographically limited track and without communication with other tracks, on which pass only the rolling stocks which are part of the system;
- have very different track layouts compared with conventional rail, namely much less complex from the point of view of the interconnections between the tracks in the stations but with winding routes and stops very close each others that do not allow to reach speeds comparable to those of conventional trains;
- the tracks are often underground and don't allow the use of communication infrastructures like GSM-R, but require communicators of small size/power arranged along the line in large numbers and to brief intervals.

The CBTC signaling system presents some aspects that derive by the ERTMS/ETCS standard and appropriately contextualized in urban applications.

Despite the differences relating to the characteristics of the track, as described above, the CBTC system has similar needs to those of a conventional rail and can include an ATC similar to that provided by the ERTMS/ETCS project, simplifying and/or refining some aspects. However, any changes must always ensure the safety properties: feature and availability comparable to those initials.

2 ATO System Analysis

The development of railway and metro signalling platforms in Europe shall comply with the CENELEC standards [14][15][16]. These are a set of norms and methods to be used while implementing a product having a determined safety-critical nature. If a company wishes to achieve a CENELEC certification for its CBTC product, the development of the product shall follow the guidelines and the prescriptions of the norms. In principle, the company can decide to treat the CBTC product as a single system, and provide certification for the system as a whole. Nevertheless, once the company has to sell a product variant, the certification process shall be entirely performed also for the variant, paying undesirable costs in terms of budget and time. Therefore, it is useful to develop each sub-system as an independent unit, and follow the CENELEC regulations for the development of such sub-system. Once each sub-system has got certification evidence according to the regulations, the certification of the whole CBTC product is made easier, since it can be focused solely on the integration aspects. Furthermore, if the customer requires only a specific sub-system (e.g., the ATC or the ATO system) to renew a part of its installation, the subsystem can be purchased without additional certification costs. The first documents typically edited for the development of a system in a CENELEC-compliant process are the Preliminary System Specification (PSS) and the System Requirements Specification (SYS-RS). The former is a document that summarizes the interfaces of the system, and the functionalities that are expected from the system: PSS is basically a drawing defining the system at block diagram level. SYS-RS is a document that precisely specifies the expected system behaviour, as well as the safety, performance, architectural and environmental constraints. Both documents are normally written in natural language. In our approach we suggest to derive the PSS directly from the detailed architecture [17].

To complete the analysis phase of the ATO system, it has been necessary to define the communication protocols. These allow to interface the ATO system with the onboard systems and with those on the ground. With the definition of the protocols documents, which define the technologies to be used for each link and the structures of the exchanged messages, it is possible to design the ATO system independently of the others.

2.1 Main challenges

Our goal is to study a protection and control system based on the CBTC technology in order to derive the onboard Automatic Train Operation (ATO) subsystem, exploiting some of the technologies of the ERTMS/ETCS project, and a wireless communication infrastructure that outperforms GSM-R limits. The development of the Automatic Train Operation (ATO) system is been influenced by the following aspects:

- adherence to reference standards of Communications-Based Train Control technology: IEEE 1474.1-2004, IEC 62290;
- compliance with CENELEC safety standards EN 50128, 50126, 50129 in order to obtain a software which meets the demands for safety;
- compliance with ERTMS/ETCS Baseline 3 (BL3) for the use of ERTMS/ETCS technologies;
- the system is part of an overall system characterized by an open and distributed architecture. This makes maintenance easier, and it also allows each sub-system to be upgraded without affecting the overall system;
- the exploitation of open standard wired and wireless network technologies provides numerous benefits ranging from the use of universally accepted protocols (Ethernet and TCP/IP) to the use of a well developed architecture and documented standards. The use of an open standard data communication system offers a stable future migration path as any of its distinct elements can be independently modified/upgraded as technology advances;
- the use of specific technologies to record and monitor all operations/events which happen when the system is active.

2.2 ATO Preliminary System Specification

It shows the internal subsystems and important interfaces to neighbouring systems. PSS can be organized into Generic Product, Generic Application and Specific Application as described in the Safety Approval Process.

The ATO System Design document has been realized starting from the analysis of the IEC 62290 and IEEE 1474.1-2004 standards and from the analysis of solutions on the rail market [18]. Regarding the analysis of the market, we have analyzed the solutions proposed by major signaling systems vendors: Bombardier, Alstom, Thales, Invensys Rail Group, Ansaldo STS, Siemens and GE Transportation.

The requirements standards have been analysed to identify the functionalities expected from a standard-compliant CBTC system (Functionality Identification), while the publicly available documents of the selected vendors are inspected to identify the CBTC architectures available in the market (Architecture Identification). Requirements standards are also employed in the Architecture Identification task to provide a common vocabulary to describe the architectures.

The main features of the ATO system that have been identified are 6: Change driving modes, Automatic speed regulation, Platform berthing control, Door control, Fault recording and reporting to ATS, Train initialization.

- **Change driving modes:** this basic function is intended to change driving modes of the train during train services between different automatic modes and from and to different manual modes for operational or fall back reasons.
- **Automatic speed regulation:** the starting, stopping, and speed regulation of the train as it travels along the track shall be automatically controlled by an ATO system so that the speed, acceleration, deceleration, and jerk rates are within specified passenger comfort limits (as defined by the authority having jurisdiction) and the train speed is below the overspeed limits imposed by ATC.
- **Platform berthing control:** an ATO system shall be capable of implementing any platform berthing control modes.
- **Door control:** an ATO system shall be capable of automatically controlling train doors (and platform edge doors, where fitted) during passenger boarding and discharging.
- **Fault recording and reporting to ATS:** failures and out-of-tolerance conditions detected by, or input to, ATO that can impact the on-time performance of the transit system or result in some other loss of specific functionality may be automatically indicated on the ATS user interface display. Any alarms shall be categorized and prioritized into critical and noncritical alarms and logged.
- **Train initialization:** ATO must have the capability to initialize all the onboard systems. In particular, it must be able to complete the initialization procedure of ATC.

Different possible architectures for a CBTC system are identified by evaluating the available information about the CBTC products on the market. Several implementations of CBTC systems are offered by different vendors. The major subsystems identified in the evaluated CBTC systems are ATC, ATS, ATO and IXL. The adopted terminology is the one provided by the CBTC standards, since the vendors use slightly different terms to refer to the same components. There are also other additional subsystems, which include, e.g., the fire emergency system, the passenger information system, and the closed-circuit television.

At the end of the first phase of analysis, we identified the operating context of the ATO system in order to define the interfaces with the external environment and the messages to be exchanged. In Figure 9 you can see the diagram that represents the operating context identified.

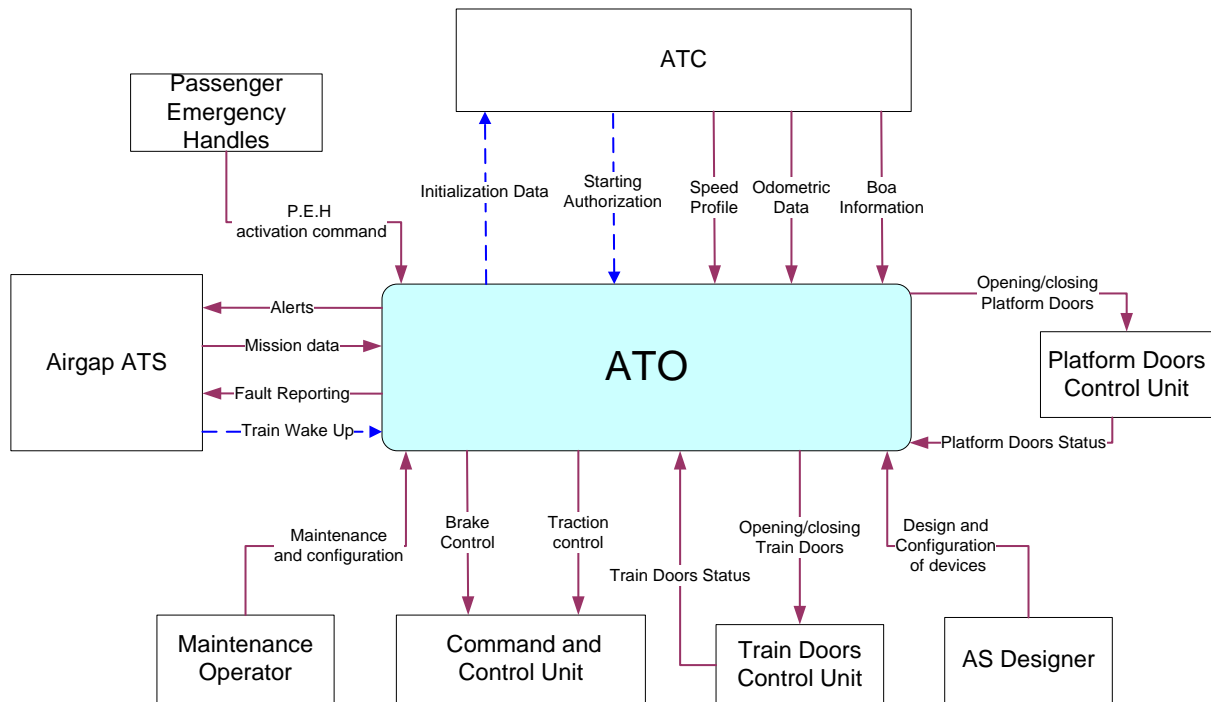


Figure 9: ATO Operating Context

The ATO system must be interfaced with the following entities:

- **ATC:** train protection system conforming to level 2 of the ERTMS / ETCS;
- **Airgap ATS:** WLAN connection between the ATO and Supervision System ATS;
- **Passenger Emergency Handles:** handle operable by the passengers on board the train for emergency braking;
- **Command and Control Unit:** device that manages the braking and the traction of the train;
- **Train Doors Control Unit:** Device for train doors opening and closing;
- **Platform Doors Control Unit:** Device for platform doors opening and closing;
- **AS Designer:** human figure that deals with the design and conFigureition of the ATO system equipment;
- **Maintenance Operator:** human figure that takes care of the maintenance/ conFigureition/ SW installation/ ATO system diagnostics.

With regard to the information exchanged between the ATO and external entities, these have been identified on the basis of the functionality associated with the ATO: for example, for automatic speed regulation the ATO needs to know the speed and current position of train (odometry data), the stops to perform (mission data), the speed profile, and at the

same time must send traction and braking commands. For each flow of information the communication medium to be used has also been identified. For communication with the ATS , it was decided to use the WLAN because it is the most widely used technology by the vendors. The same communication medium is used for the exchange of information between the wayside and onboard subsystems of ATO. For communication with the entities on board the train (ATC , Command and Control Unit, Train Doors Control Unit) has been chosen to use the MVB (Multifunction Vehicle Bus) which is the current standard for communications within rail vehicles.

Once identified the operating environment, we defined a general structure of the internal architecture of the ATO application. In Figure 10 you can see the outline architecture realized.

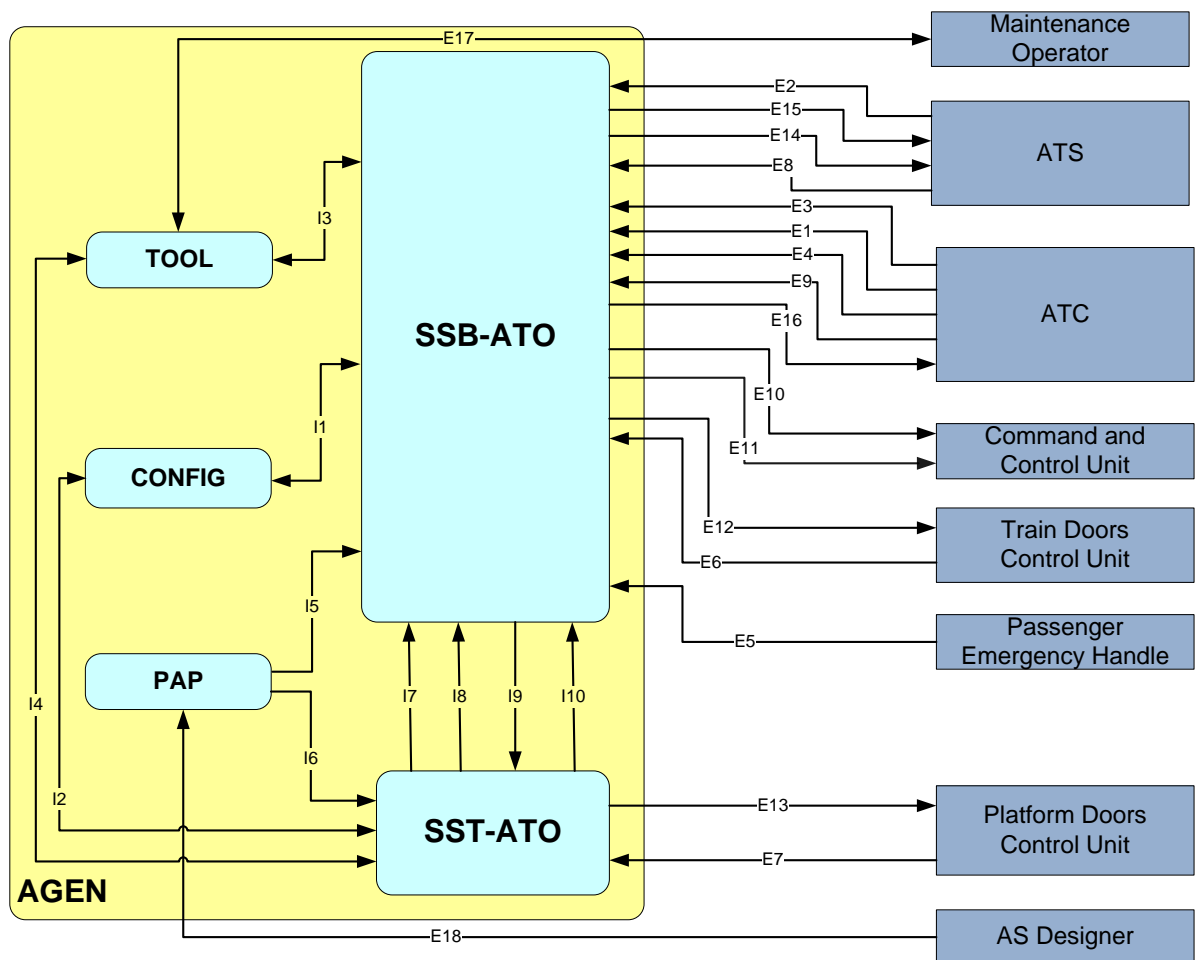


Figure 10: ATO Generic Application Architecture

The following entities define the Generic Application Architecture:

- **SSB-ATO:** onboard subsystem of ATO system that handles the automatic train management;
- **SST-ATO:** wayside subsystem of the ATO system that deals with the management of platform screen doors and tags;
- **PAP:** procedures for the installation and commissioning of the ATO system. Contains the rules for the design of the system. It interfaces with both the ATO and with SSB-SST-ATO;
- **TOOL:** set of hardware and software tools for maintenance / conFiguretion / installation / ATO system diagnostics;
- **CONFIG:** represents the configuration file for the ATO system.

2.3 ATO System Requirements Specification

For defining the requirements document for ATO system [19], we have used the information obtained from the analysis of standards, those derived from the analysis of the market solutions and design concepts contained in the ATO System Design document. Starting from the functionality associated with the ATO and from the information exchanged with external entities were extracted requirements for a railway automatic driving system.

The requirements were divided into 7 groups: technological requirements, mechanical requirements, interface requirements, functional requirements, performance requirements, RAM requirements and safety requirements. Requirements are normally written in natural language, and, following the CENELEC norms, they shall be *complete, clear, precise, unequivocal, feasible, verifiable, testable and maintainable*.

In our approach, the format of requirements is based upon four simple formats that shall be employed to write requirements. The formats are reported below:

- **FORMAT1.** *The system [shall/should] be able to < capability >.* This format is employed in case of requirements that involve mandatory (shall) or optional (should) functionalities, which are unconditional and independent from the actions of the operators. Requirements of this type are normally associated to interface functions, internal procedures, or procedures that manage internal data structures.
- **FORMAT2.** *The system [shall/should] allow the < operator > to < action >.* This format is employed in case of requirements that involve mandatory (shall) or optional (should) functionalities, which are unconditional and dependent from the actions of the operators.
- **FORMAT3.** *The system [shall/should] < action >, [when/after/before/if] < condition > {[when/after/before/if] < condition >}*. This format is employed in case of requirements that involve mandatory (shall) or optional (should) system actions that depend on one or more conditions. All conditions are considered in a

logical AND relationship. If we want to express logical OR among conditions, it is recommended to add a new requirement.

- **FORMAT4.** *[FORMAT1|FORMAT2|FORMAT3], < procedure >*. The format is a combination of one of the previous formats with a procedure. This format is employed in case of requirements that involve functionalities that have an associated procedure, or that are performed through a well-defined interface device. The format shall be used when it is useful to explain how the system is expected to perform a certain action.

The fields < capability >, < action >, < condition > and < procedure > are free-form sentences, with the only constraint of containing one verb maximum.

Before defining requirements we have defined the operating environment in which the ATO system will have to work in order to define the limits of temperature, humidity and altitude that the hardware will have to comply in accordance with EN 50155 and EN 50125-1.

Each requirement is identified by a unique code made up as follows: **ATO_SYS_nnnn** (nnnn is a sequential number unique for ATO system and the assignment is done in steps of 10 to allow subsequent insertions). Furthermore, the requirements possess a qualifier whose meaning is shown in Table 1.

Qualifier	Meaning
S	Requirement covers issues of safety integrity. S implies E (a safety requirement is essential)
E	The requirement is to be considered essential for the operation of subsystem
D	Requirement is given for future developments
O	Optional
U	Requirement is instable because some details are still being finalized
X	Requirement is eliminated but is retained in the document and is specified the reason for cancellation
V	Requirement is to be considered as an implementative constraint. V implies E (an implementative constraint is essential).

Table 1: Requirement Qualifiers

Technological requirements describe the technologies required for the installation of a component: for ATO, there are no special requirements except that the equipment must conform to the standards EN 50155 and must comply with the environmental conditions described above.

Mechanical requirements describe the mechanical components of a system: for ATO, a selector must be present for manually changing the level of automation on board the train, and tags near the stations to facilitate a more precise stop of the train.

Interface requirements define how communications between ATO and external entities must be managed: in particular they define the messages exchanged between the ATO system and external entities described in the System Design document and the information the ATO system needs to perform its tasks (doors status, signaling constraint, mission profile, TRN, etc..). For example, ATO_SYS_2010, ATO_SYS_2070 and ATO_SYS_2090 represent the two types of requirements that are part of this section. The former expresses how the ATO system should behave in communications with ATC, the latter two describe data that must be included in the mission profile that ATS must send to ATO:

- **ATO_SYS_2010** *[E] ATO system shall interface with the ATC through the virtual Driver Machine Interface (DMI) during the initialization of train.*
- **ATO_SYS_2070** *[E] ATO system shall receive a Mission Profile as specified in the document [Ref 2] and containing at least the following data:*
 - a) length of each section in which mission is divided;*
 - b) service speed for every section in which mission is divided;*
 - c) stopping points where the train has to stop;*
 - d) the departure time from each scheduled stop;*
 - e) side of door opening for each scheduled stop. The values of this data should indicate whether this is valid only for the doors of the train or even for those of platform, denoting the presence or absence of the latter.*
 - f) how long the train and platform doors (if any) must stay open.*
- **ATO_SYS_2090** *[E] The ATO system should be able to acquire the status of the train doors and platform doors (if present) from each doors control unit mounted on the train and on the ground.*

Functional requirements describe functionalities of a software system in terms of services that the system must provide, of response to specific types of input and of behaviour in particular situations: they describe the interactions between the system and its environment independently of its implementation (the environment includes the user and any other external system). They are the largest part of the requirements document since they define what the system should do and should not do. To get the definition of this type of requirements we started from the description that the standards give of the features that have been associated with the ATO system. In particular, functional requirements describe the actions to complete the initialization procedure of the ATC system, how to process the mission profile received by the ATS, how to calculate the speed profile of the service and the

signaling and how to adjust the speed based on all these data. Some examples of functional requirements of the ATO system are the following:

- **ATO_SYS_3040** [E] *The ATO system must enter and/or confirm the Driver ID, Level ERTMS/ETCS Train Date when requested by ATCC.*
- **ATO_SYS_3100** [E] *The ATO system must always accept a new Mission Profile if the system is operating at its nominal conditions.*
- **ATO_SYS_3120** [E] *The ATO system must determine the service profile based on the Mission Profile received.*
- **ATO_SYS_3170** [E] *When the train stops at a station, the ATO system will have to close the train doors and the platform doors (if any) after the opening time imposed by the Mission Profile is elapsed.*
- **ATO_SYS_3200** [E] *The ATO system should automatically control the train based on the service profile and the signaling profile by regulating traction and brake, and communicating with the Command and Control Unit.*
- **ATO_SYS_3260** [E] *The ATO system must ensure that the train speed is less than the alert speed margin, defined in ATO_SYS_4070, for each point of the track.*

Performance requirements include the requirements that relate to the number of terminals supported, the number of users who have competitive access to the system and the amount and type of information that can be manipulated at the same time. In the case of ATO, performance requirements describe the speed limits that must not be exceeded, the maximum values of acceleration and jerk and accuracy of measured train location for programmed station stop purposes. Some examples are:

- **ATO_SYS_4020** [E] *The ATO system must stop the train at the station with a maximum error of ± 30 cm if there are no platform screen doors.*
- **ATO_SYS_4040** [E] *The ATO system must generate a service speed profile such that the maximum acceleration of the train is between $+a$ and $-a$, where a is a parameter of the system configuration.*
- **ATO_SYS_4060** [E] *The ATO system must ensure that the train speed is not superior to the most restrictive condition between the service profile and the signalling profile.*

RAM requirements identify three key characteristics in the field of maintenance of a system or apparatus. RAM is an acronym for Reliability, Availability, Maintainability. Reliability of a system is the probability that the system operates in predetermined ways, such as by operating specifications, for a given period of time, according to the specific operating conditions; Availability of a system is the probability that the system, at a given instant, is capable of performing predetermined functions, such as by operating specifications, under the operating conditions prescribed, assuming that have been assured the necessary maintenance interventions; Maintainability is the property of a system to be maintained,

defined as the probability that an action of active maintenance can be performed during a given time interval, in the given conditions, through the use of prescribed procedures and facilities.

Security requirements define the security features of the system. In the case of the ATO, the only identified requirement was that each alarm/emergency that occurs on the train must be recorded and notified to the ATS.

2.3.1 ATO-ATS Communication Protocol

ATS-ATO communication protocol ensures the interaction and exchange of information between ATS and ATO. The ATS-ATO communication network is composed of several distributed nodes (ATO) and centralized entity (ATS), which supervises the movement of trains. The communication that is obtained through this protocol must respect certain fundamental characteristics: the first is that communication must be bidirectional to allow sending and receiving of data by both entities, the second is that the information exchanged does not need to be on time (sending and receiving of information does not necessarily have to respect tight deadlines) [20].

2.3.2 ATO-TRAIN Communication Protocol

The ATO-TRAIN communication protocol is necessary to allow the ATO to communicate with the train interface. This was necessary because in a train with driver, interaction takes place through levers and buttons but with an automatic guidance system these can not be used. For this reason it was necessary to define a communication protocol able to transfer commands from the ATO at the train interface (traction, braking, doors) and return the train status. The ATO-Train protocol is defined to allow the exchange of information between the automatic driving system and the TRAIN interface. ATO needs to send to TRAIN interface commands that would be implemented manually in presence of the driver: commands for activation of the ATC protection system and of the desk, braking / traction controls and commands for train doors' opening/closing. Similarly, the TRAIN interface shall send to ATO all the information needed to identify the status of the train: informations that must be reported are the activation status of ATC and of desk, the braking / traction level applied and the status of each port of the train.

2.3.3 ATO-ATC Communication Protocol

Communication with ATC requires the use of UDP protocol: this protocol is the same as that used for communications between ATC and DMI to display information on the screen according to ERTMS standard. The ATO must replace both the driver responding to the

requests of ATC and the DMI behaving like a virtual DMI. The communication protocol between ATC and ATO is substantially different from the one between ATS and ATO due to the different protocol used (UDP instead of TCP) but also from the point of view of the communication itself. In ATS-ATO communication there is no predefined messages sequence (messages are sent when needed), instead in the communication with ATC the ATO can send a message only in response to one sent by ATC. Messages sent by ATC have a timing of 100 ms and the ATO must reply before next message sent by ATC, otherwise it is considered inactive.

2.4 Initialization Phase

As previously mentioned, one of the basic requirements of the system that will be implemented is the integration with the standard ETCS level 2. Integration of the CBTC system with an ETCS system will create a system for metropolitan edge that meets the highest levels of safety and innovation currently present worldwide. The integration of an ETCS level 2 requires that the ATO undertakes all the functions that would be performed by the driver: these include the initialization phase that must be completed in order to start the on board system.

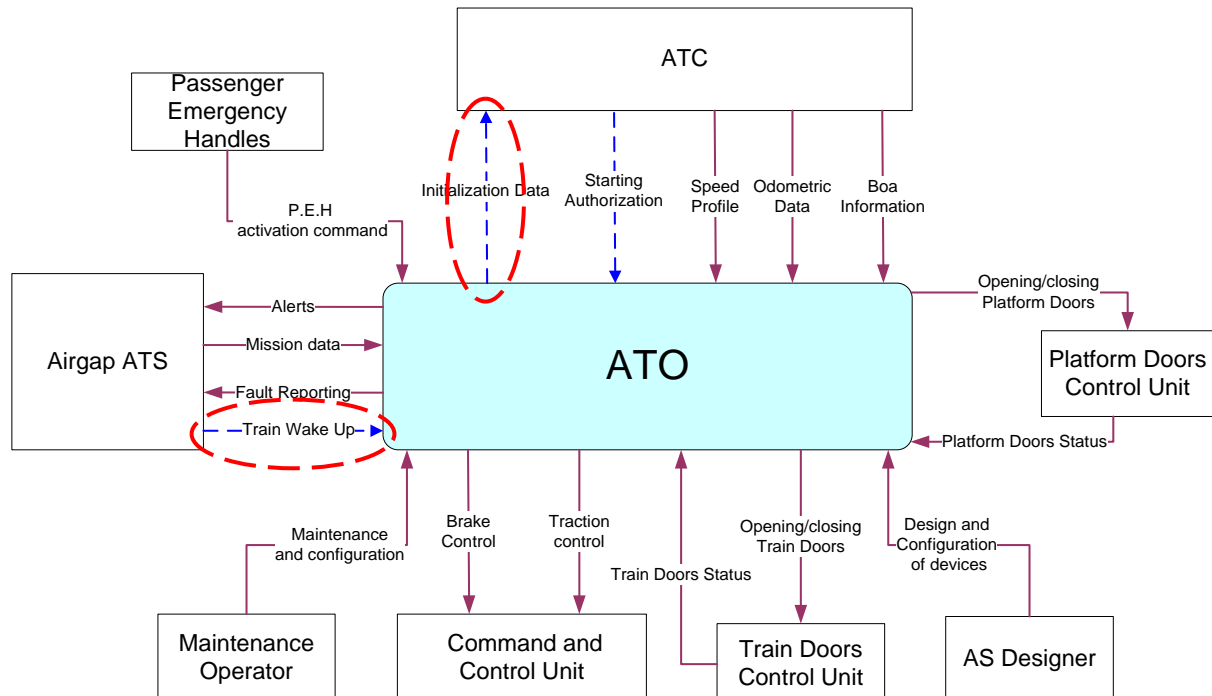


Figure 11: CBTC System Architecture

As it can be seen from the figure, for the initialization phase ATO must exchange data with both ATS and ATC (which implements an ETCS system). Communication with ATS includes the command WakeUp to awake the ATO and the data of the mission needed to complete the procedure Start of Mission and to program the stops to be carried out. ATS-ATO communication protocol is defined in the document [20].

Communication with ATC involves an exchange of information vital to the management of train: ATO obtains the necessary data for the train running (current speed, distance traveled, speed profile to follow to meet the criteria of safety, etc.) from ATC, at the same time ATO must submit the requested information during the procedure Start of Mission to ATC, essential for starting and configuring the ATC system. In fact, the ATO must totally replace the driver but also the DMI, that is the interface device between the driver and the ATC. This feature of the ATO is due to the fact that the communication between ATC and ATO must comply with the communication protocol described in the document [21].

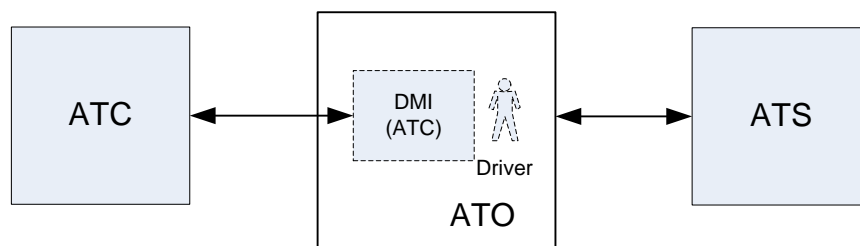


Figure 12: Components involved in Initialization

As mentioned earlier, the ATO must supervise all those operations that would be made by the human operator, including the switching on of all on-board systems. The initialization of the systems on board the train takes place in two well defined cases: the first time in which the system is switched on and after a standby phase. This functionality can be divided into two phases of which only the first presents differences between the two cases. When the system is turned on for the first time the steps to take are as follows:

1. Initially all communication protocols (to ATC, ATS, interface of the train) must be started and the ATO must "present" itself to ATS and waits for a signal by the ATS itself.

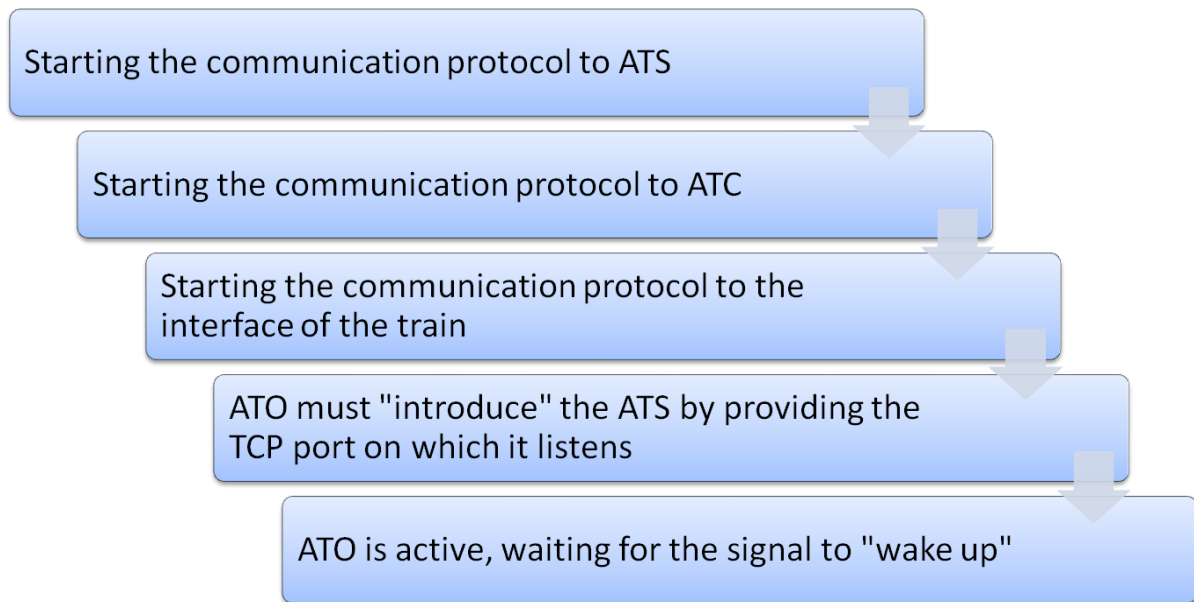


Figure 13: Preliminary operation of Initialization phase

2. When ATO receives the signal "Wake Up" by the ATS must activate the ATC system, activate a desk and carry out the procedure Start of Mission.

After these two phases the onboard system is ready to move the train and to complete its "mission".

When the train is located in storage at the end of service, the onboard system is asleep in a standby phase in which most of the systems are turned off and only the communication with ATS remains active to allow the reception of the command of "awakening" of the system. Compared to the first case described previously, there are fewer operations required to activate the onboard system: the communication with the ATS is already active and is not necessary that ATO "presents" itself, then only the communication protocols with ATC and the interface of the train must be switched on.

2.4.1 Procedure Start of Mission

Procedure Start of Mission is described in Chapter 5 of Subset 026 of ETCS standard [8]. The procedure Start of Mission is started by the driver once the train is awake or once a mission is ended. At the beginning of the Start of Mission procedure, the data required may be in one of three states:

- a) "valid" (the stored value is known to be correct)
- b) "Invalid" (the stored value may be wrong)
- c) "Unknown" (NO stored value available)

This refers to the following data: Driver ID, ERTMS/ETCS level, RBC ID/phone number, Train Data, Train Running Number, Train Position (see 3.6.1.3).

Procedure

The following is the table of requirements for “Start of Mission” procedure. Figure 14 show the flowchart of SoM procedure. The ID numbers in the table are used for the representation of the procedure in form of a flow chart.

ID #	Requirements
S0	The Start of Mission procedure shall be engaged when the ERTMS/ETCS on-board equipment is in Stand-By mode with a desk open and NO communication session is established or is being established.
S1	<p>Depending on the status of the Driver-ID, the ERTMS/ETCS on-board equipment shall request the driver to enter the Driver-ID (if the Driver-ID is unknown) or shall request the driver to revalidate or re-enter the Driver-ID (if the Driver-ID is invalid).</p> <p>The ERTMS/ETCS on-board equipment shall offer the driver the possibility to enter/re-validate (depending on the status) the Train running number.</p> <p>The ERTMS/ETCS on-board equipment shall also offer the driver the possibility to set/remove a Virtual Balise Cover.</p> <p>Once the Driver-ID is entered or revalidated (E1) (possibly further to the Train running number entry/revalidation and/or to Virtual Balise Cover setting/removal), the process shall go to D2</p>
D2	<p>If both the stored position and the stored level are valid, the process shall go to D3</p> <p>If the stored position or the stored level is “invalid” or “unknown”, the process shall go to S2</p>
D3	<p>If the stored level is 2 or 3, the process shall go to D7</p> <p>If the stored level is 0,1 or NTC, the process shall go to S10</p>
D7	<p>If at least one Mobile Terminal is registered to a Radio Network, the process shall go to A31</p> <p>If NO Mobile Terminal is registered to a Radio Network, the process shall go to A29</p>

ID #	Requirements
S2	<p>If the status of the Level data is "unknown", the ERTMS/ETCS on-board equipment shall request the driver to enter it.</p> <p>If the status of the Level data is "invalid", the ERTMS/ETCS on-board equipment shall request the driver to re-validate or re-enter the ERTMS/ETCS level.</p> <p>If the entered / re-validated level is 2 or 3, the process shall go to S3</p> <p>If the entered / re-validated level is 0, 1 or one of proposed NTC level(s) (see 3.18.4.2 for the levels the driver is allowed to select), the process shall go to S10</p>
S3	<p>The ERTMS/ETCS on-board equipment shall offer the possibility to the driver to re-enter the Radio Network ID. If the driver elects to do so, the on-board equipment shall acquire an alphanumeric list of available and allowed networks, based on a request to the Mobile Terminal(s) and:</p> <ul style="list-style-type: none"> • If this list is empty (E3) the process shall go to A29 • If the driver selects a new Radio Network ID from the proposed list, the registration of the Mobile Terminal(s) to this new Radio Network shall be ordered and the status of the RBC-ID/phone number shall be immediately set to "unknown". <p>If at least one Mobile Terminal is registered to a Radio Network, the ERTMS/ETCS on-board equipment shall offer the following options to the driver for the RBC-ID/phone number:</p> <ul style="list-style-type: none"> • Only if the status of the RBC-ID/phone number is "invalid": order the ERTMS/ETCS on-board equipment to use the last stored RBC-ID/phone number • Order the ERTMS/ETCS on-board equipment to use the EIRENE short number (trackside call routing function) • Enter the RBC-ID/phone number (if its status is "unknown"), or revalidate/re-enter it (if its status is "invalid"). <p>Once the driver has selected the first or second option or once data is validated (E5), the process shall go to A31</p>

ID #	Requirements
A29	<p>The ERTMS/ETCS on-board equipment shall inform the driver that the Radio Network registration has failed</p> <p>This condition leads to S10 (the driver has to unlock the situation to continue e.g. selection of new level)</p>
S10	<p>The ERTMS/ETCS on-board equipment shall offer the possibility to the driver to select SH, NL, or to select Train Data Entry.</p> <ul style="list-style-type: none"> • If the driver selects SH (E12), the process shall continue in the same way as the procedure “Shunting initiated by the driver”. If, in level 2 or 3, the RBC rejects the request for Shunting (E13), the process shall go back to S10. • If the driver selects NL (E10) then the ERTMS/ETCS on-board equipment shall immediately switch to Non Leading mode (refer to SRS chapter 4, transition between modes: transition [46]). The mission starts in NL mode (if level is 2 or 3, the ERTMS/ETCS on-board equipment also reports the change of mode to the RBC). • If the driver selects Train Data Entry (E11), the process shall go to S12 • Following E10, E12, if the position is still invalid, the ERTMS/ETCS on-board shall delete the train position data (new status: “unknown”)
S12	<p>The ERTMS/ETCS on-board equipment shall request the driver to enter/revalidate the Train Data that requires driver validation</p> <p>.</p> <p>Once Train Data is stored and validated (E16), the process shall go to D12</p>
D12	<p>If Train running number is valid, the process shall go to D10</p> <p>If Train running number is “unknown” or “invalid”, the process shall go to S13</p>
S13	<p>If the status of the Train running number is "unknown" or “invalid”, the ERTMS/ETCS on-board equipment shall request the driver to enter/re-validate the Train running number now.</p> <p>Once Train running number is entered/re-validated (E18), the process shall go to D10.</p>

ID #	Requirements
D10	<p>When the validated level is 2/3, the process shall go to D11</p> <p>When the validated level is 0,1 or NTC, the process shall go to S20</p>
D11	<p>When the session is open, the process shall go to S11, otherwise it shall go to S10</p>
S11	<p>The ERTMS/ETCS on-board equipment shall send Train Data to the RBC.</p> <p>When the RBC acknowledges Train Data (E14), then the ERTMS/ETCS onboard equipment shall go to the step S20.</p>
S20	<p>The ERTMS/ETCS on-board equipment shall offer the possibility to the driver to select "Start"</p> <ul style="list-style-type: none"> a) When the validated level is NTC and the driver selects "start" (E20), the process shall go to S22 b) When the validated level is 0 and the driver selects "start" (E21), the process shall go to S23 c) When the validated level is 1 and the driver selects "start" (E22), the process shall go to S24 d) When the validated level is 2 or 3 and the driver selects "start" (E24), the process shall go to S21
S21	<p>The ERTMS/ETCS on-board equipment shall send an MA request to the RBC and wait.</p> <p>If an SR authorisation is received from RBC (E26), the process shall go to S24</p> <p>If an MA allowing OS/LS/SH is received from RBC (E27), the process shall go to S25</p> <p>If an MA allowing FS is received from RBC (E29), the mission starts in Full Supervision mode (refer to SRS chapter 4, transitions between modes: transition from SB to FS)</p>

ID #	Requirements
S22	<p>The ERTMS/ETCS on-board equipment shall request an acknowledgement from the driver for running under supervision of the selected National System. When the driver acknowledges (E30) , the mission starts in SN mode (refer to SRS chapter 4, transitions between modes).</p> <p>Following E30, if the position is still invalid, the ERTMS/ETCS on-board shall delete the train position data (new status: “unknown”)</p>
S23	<p>The ERTMS/ETCS on-board equipment shall require an acknowledgement from the driver for running in Unfitted mode. When the driver acknowledges (E31), the mission starts in Unfitted mode (refer to SRS chapter 4, transitions between modes: transition from SB to UN)</p> <p>Following E31, if the position is still invalid, the ERTMS/ETCS on-board shall delete the train position data (new status: “unknown”)</p>
S24	<p>The ERTMS/ETCS on-board equipment shall require an acknowledgement from the driver for running in Staff Responsible mode. When the driver acknowledges (E32), the mission starts in SR mode (refer to SRS chapter 4, transitions between modes: transition from SB to SR)</p> <p>Following E32, if the position is still invalid, the ERTMS/ETCS on-board shall delete the train position data (new status: “unknown”)</p>
S25	<p>The ERTMS/ETCS on-board equipment shall require an acknowledgement from the driver for running in On Sight/Limited Supervision/Shunting mode. When the driver acknowledges (E33), the mission starts in On Sight/Limited Supervision/Shunting mode (refer to SRS chapter 4, transitions between modes: transition from SB to OS, LS or SH)</p>
A31	<p>The ERTMS/ETCS on-board equipment shall open the session with the RBC.</p>
D31	<p>If the opening of the session is successful, the process shall go to D32</p> <p>If the opening of the session has failed, the process shall go to A32</p>

ID #	Requirements
A32	<p>The driver shall be informed when the on-board equipment fails to open a radio session.</p> <p>Opening of a radio session has failed if</p> <ul style="list-style-type: none"> • NO connection to the RBC can be established (see section 3.5.3.7) OR • The ERTMS/ETCS on-board equipment, based on the system configuration reported by the RBC, decides that compatibility is not ensured and terminates the communication session <p>This condition leads to S10 (The driver has to unlock the situation to continue e.g. selection of new level).</p>
D32	<p>If the stored position is valid, the process shall go to A33</p> <p>If the stored position is invalid, the process shall go to A34</p>
A33	<p>If the train position data stored in the on-board equipment is of status "valid", the train position, marked as "valid" shall be transmitted to the RBC via the "SoM position report" message.</p> <p>This condition leads to S10.</p>
A34	<p>If the train position data stored in the on-board equipment is of status "invalid" or "unknown", the train position, marked as "invalid" or "unknown" shall be transmitted to the RBC via the "SoM position report" message.</p> <p>The process shall then go to D33</p>
D33	<p>When the position report marked as "invalid" is received by the RBC, this latter shall check whether it can validate this position report.</p> <p>If the position report can be validated by the RBC, the process shall go to A35</p> <p>Otherwise, if the position report was marked "unknown", or the "invalid" position report cannot be validated by the RBC, the process shall go to D22</p> <p>Note: How the RBC is able to validate the position report is a national issue, out of the scope for this specification</p>

ID #	Requirements
A35	<p>The RBC shall inform the ERTMS/ETCS onboard equipment that the reported position is valid.</p> <p>When this message is received by the ERTMS/ETCS on-board equipment, the status of the position shall be set to "valid"</p> <p>The process shall go to S10.</p>
D22	<p>If the reported train position is "unknown", or the RBC is not able to confirm a reported "invalid" position, the RBC shall nevertheless decide whether it accepts the train or not.</p> <p>If yes, the process shall go to A23</p> <p>If NO, the process shall go to A38</p> <p>Note: How the RBC assumes responsibility for the train is a national issue, out of the scope for this specification</p>
A23	<p>The RBC shall inform the ERTMS/ETCS on-board equipment that it accepts the train although the on-board has NO "valid" position information.</p>
A24	<p>When the ERTMS/ETCS on-board equipment is informed that the train is accepted without valid position data, it shall delete the train position data (new status: "unknown")</p> <p>This condition leads to S10.</p>
A38	<p>The RBC shall inform the ERTMS/ETCS on-board equipment that it rejects the train</p>
A39	<p>When the ERTMS/ETCS on-board equipment is informed that the train is rejected, it shall delete the train position data (new status: "unknown") and shall terminate the session with the RBC.</p> <p>The process shall then go to A40</p>
A40	<p>The ERTMS/ETCS on-board equipment shall inform the driver that the train is rejected</p> <p>This condition leads to S10 (the driver has to unlock the situation to continue e.g. selection of new level).</p>

Table 2: Start of Mission procedure

The SoM procedure shall end as soon as at least one of the following conditions is fulfilled:

- Transition to any mode other than SB
- The desk is closed

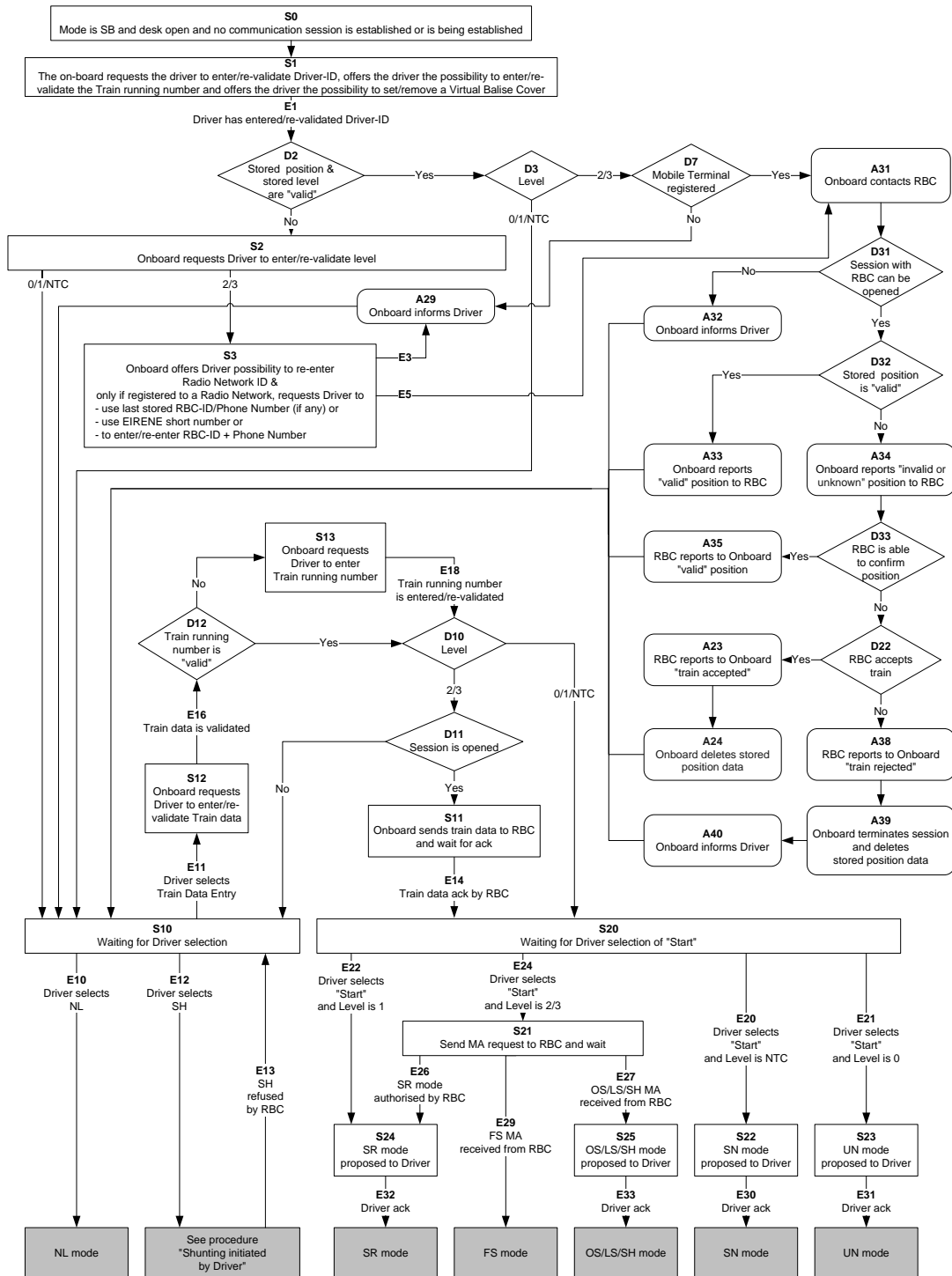


Figure 14: Flowchart for Start of Mission Procedure

In our case we do not need all steps defined by the Start of Mission procedure but have been taken into account only the states in which must be included information essential to ETCS. In particular, the data to be inserted / confirmed are the following:

- **Insertion / confirmation of data**
 - Driver ID (confirmation of a default value, in case of GoA4)
 - Level (confirmation of level 2 -pre-set value)
 - Train Date (confirmation of default settings)
- **Insertion of the Train Running Number**
 - This is the "operating number " (must be supplied by ATS)
- **(ATC-RBC communications - waiting for ACK by RBC)**
- **RBC ACK -> Request of the command "Start"**
 - The START command must be sent to the ATC system
- **ATC sends request for MA to RBC**
 - After receiving the MA by RBC, the ATC system enters in FULL SUPERVISION and train can begin the mission.

3 ATO System Modelling

Within TRACE-IT project, in order to develop the on-board component of a CBTC- ATC system, we have decided to adopt a **model-based/model driven approach**, usually referred as "Model-Based Systems Engineering" - **MBSE**.

3.1 Model Based Systems Engineering

Model Drive Approach is a software development methodology that aims to enhance the connection between the system model and its final realization, increasing the coherence among the analysis and the implementation phase. This approach is more related to specific domain concepts and behavioral aspects [22] rather than algorithms or concepts of computation. Model is the central element in this approach, thus you can represent and specify a system at various level of granularity such as the operational, functional, and technical aspects. Modeling the system helps to manage its complexity, since each model and diagram provides an abstracted view and definition of the system (or part of it). Moreover MDA always keeps the model updated and related to the final system during all the phases of software development process, and even when the system is complete and maintained.

The model-driven development is characterized by the strong use of graphical notations that overshadow the writing lines of code. In this regard the principal languages of reference are UML (currently version 2.0) and SysML, which are able to simplify the complexity of a system making it easily understandable to the different actors involved in the its development.

The MDA, even different from classic programming paradigms adopted in software development, introduces great advantages in software system design and development, increasing also the compatibility between the various components. In fact you can describe system's behavior in a very detailed way, with a granularity almost comparable to that of a classic programming languages. This entails:

- the possibility to generate part or all the code required by the system after entering a sufficient amount of information to describe the model in a complete manner, keeping the consistency between system model and the implemented source code.
- the possibility to execute the model from a functional point of view, ie simulate the system's behavior (desired or not).
- the possibility to apply validation and verification techniques that allow to check the correctness of the system.

- the possibility to automatically extract the code documentation; this allows the designers to concentrate on the logic and the overall architecture of project.

Today, there are some technologies and tools that enable us to develop code without writing code, take for instance Artisan Studio, Stateflow or Eclipse Modeling Framework. According to the features described above, in this project we have used as a development tool **IBM Rational Rhapsody** (versions **Designer for Systems Engineers and Developer for C++**).

3.2 IBM Rational Rhapsody

IBM Rational Rhapsody Designer for Systems Engineers is an MBSE (Model-Based System Engineering) environment that allows to build and manage models on which to base the implementation of a system, and the operations of verification and validation of the same in the early stage of development. This solution supports major reference standards, Unified Modeling Language (UML) and Systems Modeling Language (SysML), with all their formalisms and the main features [23].

Rational Rhapsody Designer for System Engineers helps design teams to manage development challenges of complex products. In addition, it introduces a number of advantages for development teams, supporting several useful features for project management of medium to large complexity, such as:

- Design and requirements traceability - uses SysML and UML in order to facilitate requirements analysis, to execute business studies and to design behavioural and structural systems aspects.
- Graphical development - to graphically represent the system using industry standards as SysML and UML, or Domain Specific Languages (DSL) as AUTOSAR.
- Management of models and simulations - allows to validate in advance the behavioural aspects that characterize a model.
- Code Generation - allows the automatic generation of code in different programming languages, starting from the model and taking into account the behaviour shown into the statecharts.
- Documents generation - uses several tools ranging from simple report RTF generator to fully customizable tools, as Rational Publishing Engine or Rational Rhapsody ReporterPLUS.
- Collaboration of teams - helps teams to collaborate in order to manage the development complexity in coherent projects in different environments.
- Life cycle and additional software support - integrates with other IBM Rational products to support the whole development of life cycle of software systems. Moreover, it is possible to extend Rational Rhapsody Designer for System Engineers features with optional additional software.

One of the main features that led us to choose the Model Driven Development approach and the Rational Rhapsody tool is certainly the automatic code generation from the model. The IBM development environment allows you to organize the various components (such as packages, use cases, classes and all UML diagrams) that compose the model of the system in a hierarchical structure, and provides the ability to partition a system into small subsystems. Then, the tool is able to operate an automatic preliminary check on the created model in order to find potential problems before the code generation (this operation can also be started manually with predefined or custom controls). At the end Rational Rhapsody generates code, for the whole project configuration or only for selected classes, starting from the UML Model. The inputs for code generator are the model and the various properties for setting the code generation. While, the outputs of the code generator are the source files in a destination programming language: the specifications files, the implementation files and makefile.

Another important feature concerns the project documents generation. Rational Rhapsody has the internal reporting tool in order to create reports for inner use, for example as reference for debugging a model. A report can be configured with information concerning only selected items or all the elements of an active component, using standard or custom templates. The application Rational Rhapsody ReporterPLUS can create Microsoft Word, Microsoft PowerPoint, HTML, RTF documents, or text reports from Rational Rhapsody templates. The produced file can be saved and then shown in any program able to read the format of the report. Rational Rhapsody ReporterPLUS creates documents exploiting the following techniques:

- extraction of text and diagrams from a model created in IBM Rational Rhapsody.
- addition of text and diagrams from the model and images to the document.
- addition of boilerplate text specified in the Rational Rhapsody ReporterPLUS template to the document.
- Formatting the document according to the formatting commands in the Rational Rhapsody ReporterPLUS template, as well as the specifications in a Word template (.dot file), a PowerPoint template (.pot file), or an HTML style sheet (.css file). Using a .dot, .pot, or .css file is optional. HTML tags to format HTML documents can also be used.

Rational Rhapsody is also equipped with a testing environment which is based on three main components: Automatic Test Architecture Generation, Automatic Test Case Execution and Automatic Test Case Generation. Rhapsody ATG is a test case generation tool using standard Unified Modeling Language (UML) design notations. Using ATG, you can automatically generate test suites and perform test execution applications developed.

3.3 Package Init Design

This package contains the software subsystem for:

- the initialization phase of train allowing it to start its mission;
- the initialization of all other components of ATO system.

The mission specifies the service profile that allows the train to make the stops in pre-established stations. The train initialization includes the following operations: activation of ATC/ATC subsystem, activation of desk and execution of procedure Start of Mission provided by ERTMS / ETCS standard (the reference regulations of this procedure is the subset-v330 026-5). All these operations are sequentially performed in the order with which they have previously been listed. This package includes an only class which is identified by block **Initialization_Manager**, which has the task of managing the initialization phase of train in all its intermediate steps.

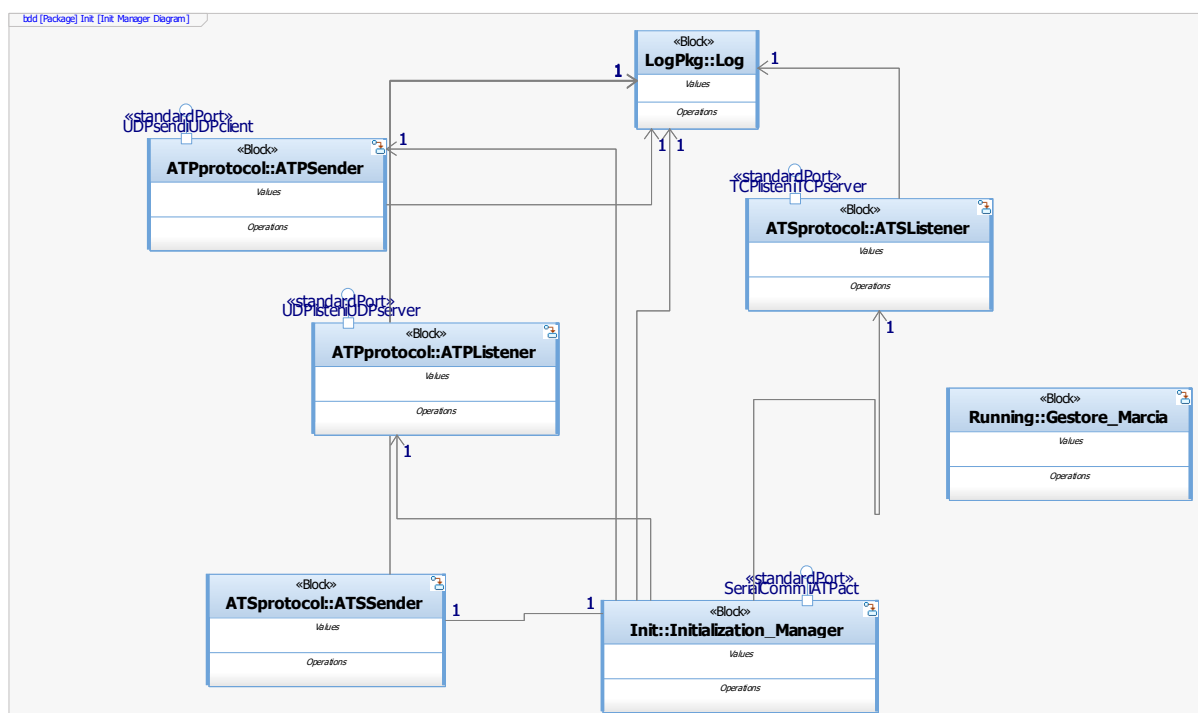


Figure 15: Init block definition diagram

As shown in the diagram in Figure 15, the block interacts with others classes of packages ATSpool, ATCprotocol and LogPkg. Use of ATSpool allows Initialization_Manager to exchange messages with ATS subsystem; in the same way the communication with ATC/ATC subsystem is managed by Initialization_Manager through ATCprotocol. The message exchange between ATO and ATS is governed by ATS-ATO communication protocol [20],

while the communication between ATO and ATP/ATC is governed by DMI ETCS-CPU32 protocol [21]. Finally, LogPkg is used to recording all events/operations relative to the class Initialization_Manager. This class manages the initialization phase of train in all its intermediate steps. So Initialization_Manager executes, as a first step, the activation of ATP/ATC subsystem, then the activation of desk, and finally the procedure Start of Mission defined by subset-v330 026-5 of ERTMS / ETCS standard [8]. At the end of the latter, Initialization_Manager is able to confirm the operating mode with which the train will start its mission. The procedure Start of Mission includes a number of steps executed according to the flowchart shown in Figure 5. These steps are realized through interaction between the driver and the DMI (driver machine interface) in an ERTMS/ETCS level 2 system. So, the ATO subsystem will replace the driver in order to perform what requested by the procedure. To achieve this, Initialization_Manager implements a "virtual DMI" plus a "virtual Driver". The involved subsystems are ATO and ATC, which are interested by a message exchange complying with DMI ETCS-CPU32 protocol. Messages coming from ATC/ATC are managed by Initialization_Manager in a specific state in order to create the appropriate response message. The latter communicates to ATP/ATC the status of shown screen. A message coming from ATC, implying a variation of screen to be displayed, leads Initialization_Manager in a new state; if instead the screen is the same of that commanded by the previous message we have a transition to a new sub-state within current state.

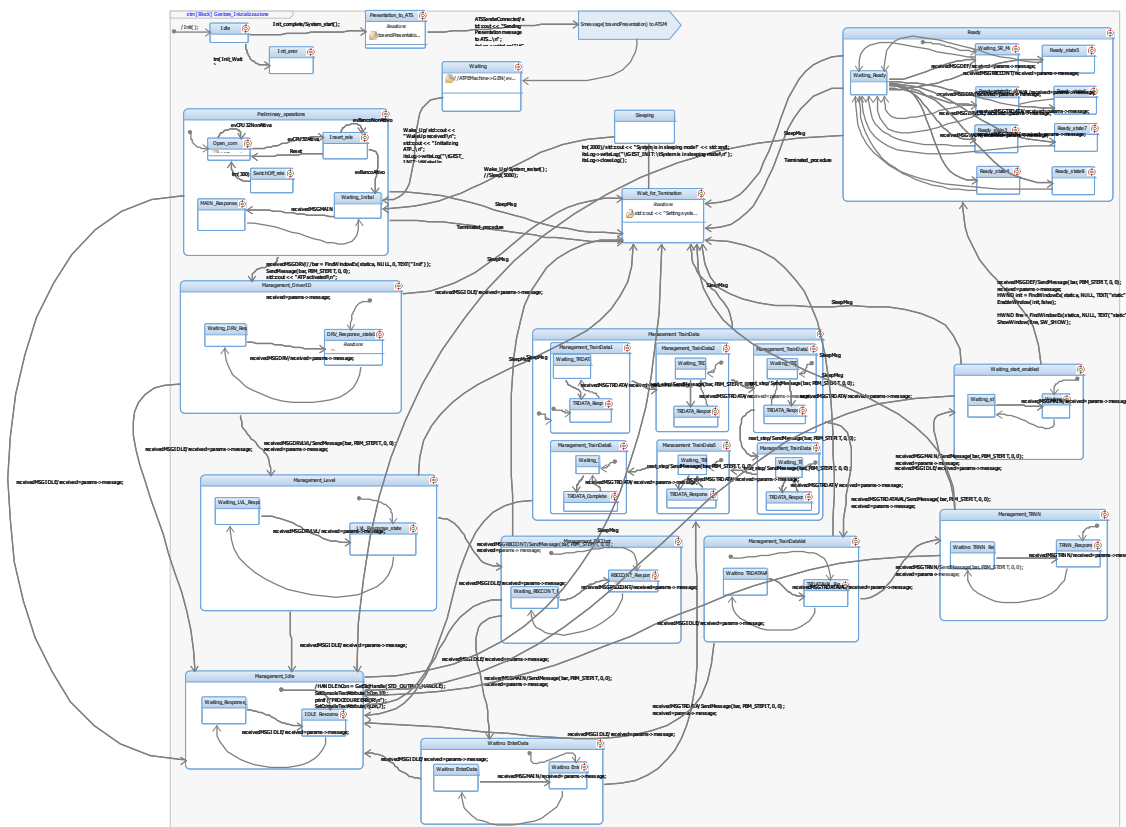


Figure 16: Statechart of Initialization_Manager

Therefore the reception of a message sent by ATP/ATC is considered as an external event that triggers a specific transition and modifies the current state (or sub-state) of Initialization_Manager.

The associate statechart to this class is shown in Figure 16. This diagram allows us to define the behaviour of Initialization_Manager specifying the possible states reached during its life cycle, and the reaction to external events in terms of changes in state and/or actions carried out. The states reached by Initialization_Manager are the following:

- **Idle:** previous state to initialization of all other components of ATO system. Preliminary operation is reading the configuration file (ato.config) in order to set the following parameters: ATS IP address (ATSaddress), ATS TCP port (ATSport), TCP listening port (TCPListen), ATC IP address (ATCaddress), ATC UDP port (ATCport), UDP listening port (UDPListen), Driver ID, ERTMS/ETCS level (level), axle load (Axle_Load), train category (Cat_Train), max train speed (Speed_Train), train length (Length_Train), brake percentage (PBrake_Train), Train Running Number (TRNN) and serial port (SERIALport).

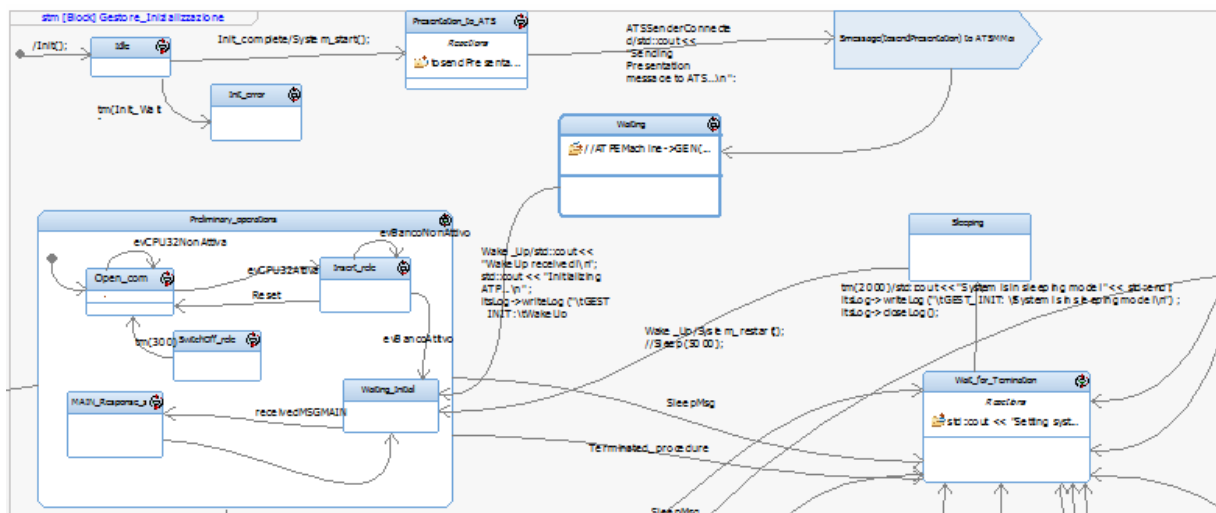


Figure 17: Sub-states of Preliminary operations state

- **Presentation to ATS:** start of the message exchange between ATO and ATS. Initialization_Manager sends to ATS a presentation message containing number port on which will take place communication ATS-ATO.
- **Preliminary operations:** Initialization_Manager executes the following operations:
 - activation of ATC/ATC subsystem through relay;
 - activation of desk through relay;
 - response to first type of message exchanged by ATO e ATC.

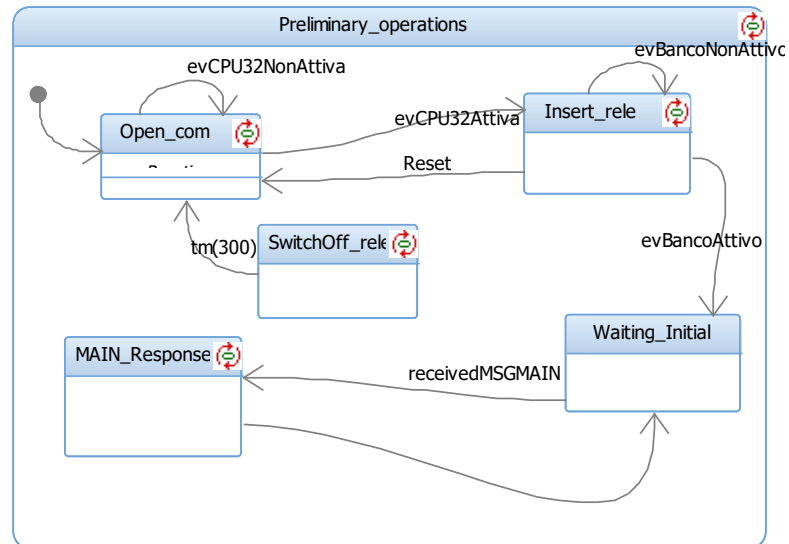


Figure 18: Sub-states of Preliminary_operations state

- Management DriverID: Initialization_Manager executes the first step of procedure Start of Mission. This step consists in Driver ID entry with value extracted from configuration file.

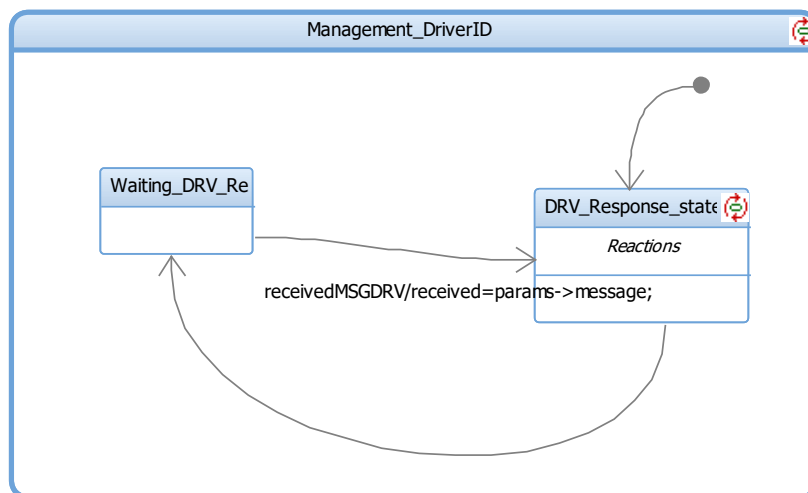


Figure 19: Sub-states of Management_DriverID state

- Management Level: Initialization_Manager executes the next step of procedure Start of Mission. This step consists in ERTMS/ETCS level entry with value extracted from configuration file.

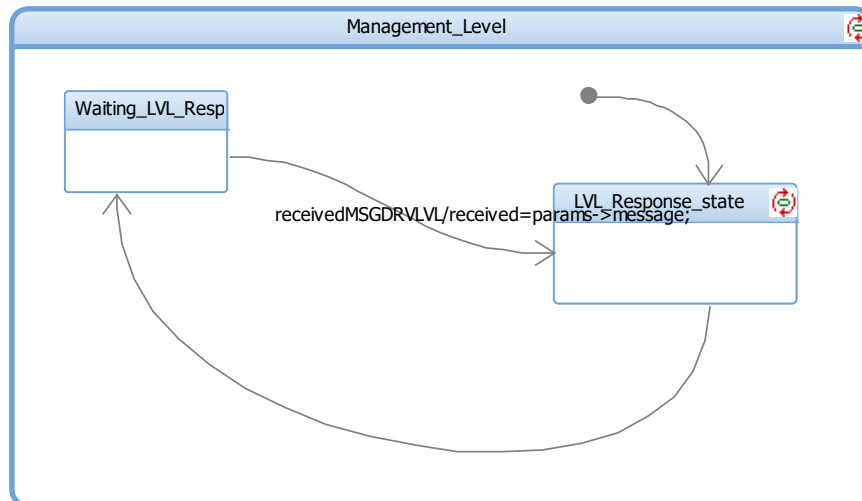


Figure 20: Sub-states of Management_Level state

- **Management_RBCCont:** Initialization_Manager executes the next step of procedure Start of Mission. This step consists in opening the session with RBC (Radio Block Centre). To do this, Initialization_Manager makes contact with last stored RBC-ID/Phone Number.

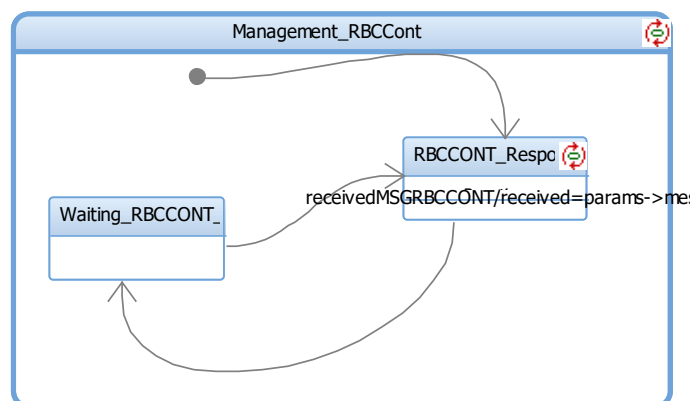


Figure 21: Sub-states of Management_RBCCont state

- **Waiting_EnterData:** waiting state in which Initialization_Manager selects the Train Data Entry option in order to enter or revalidate the train data in the next step of procedure Start of Mission.

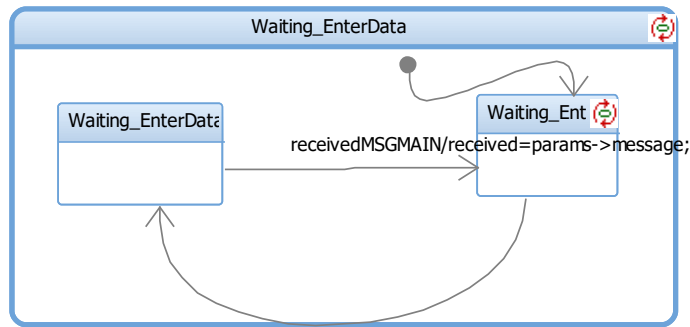


Figure 22: Sub-states of Waiting_EnterData state

- **Management_TrainData:** Initialization_Manager executes the next step of procedure Start of Mission. This step consists in train data entry with values extracted from configuration file.

As shown in Figure 23, train data are sequentially entered by Initialization_Manager with this sequential order:

1. axle load (Axle_Load)
2. train length (Length_Train)
3. max train speed (Speed_Train)
4. brake percentage (PBrake_Train)
5. train category (Cat_Train)

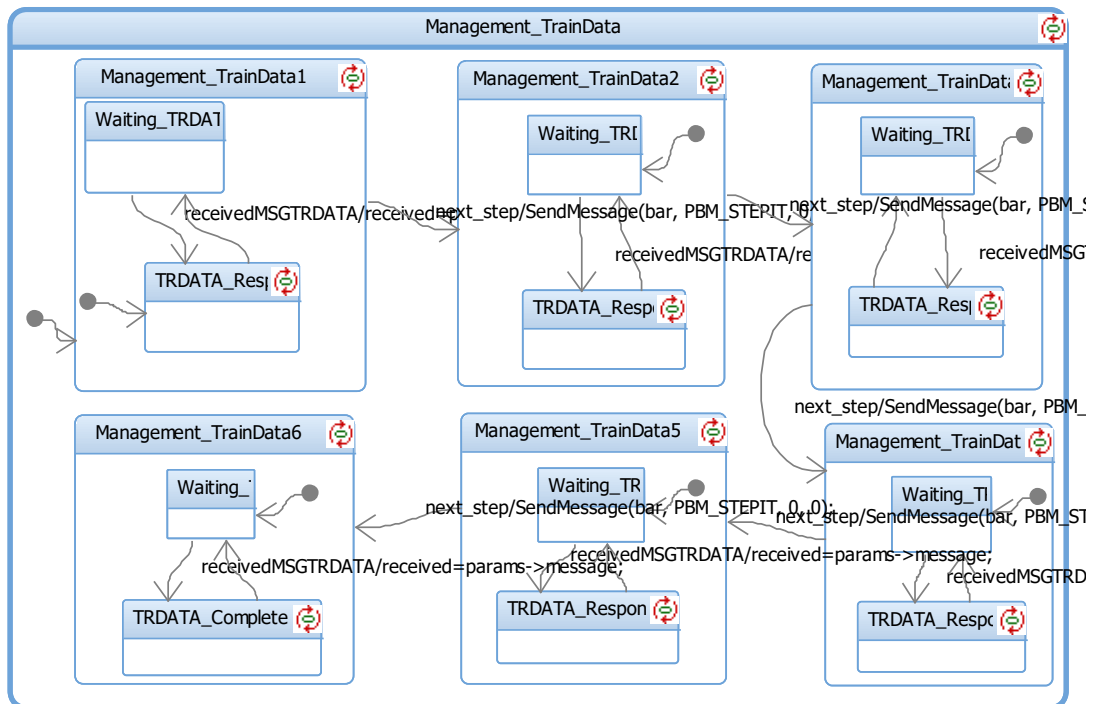


Figure 23: Sub-states of Management_TrainData state

- Management_TrainDataVal: state in which Initialization_Manager validates train data previously entered.

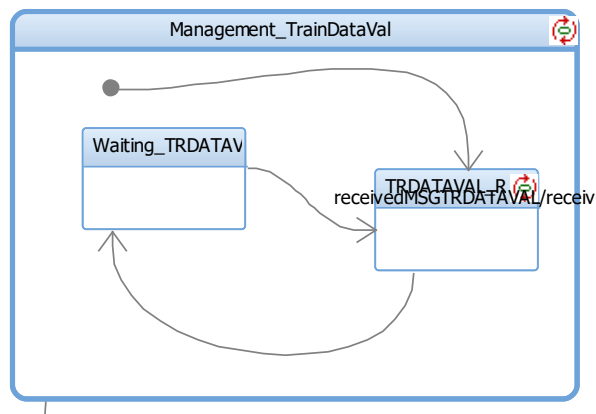


Figure 24: Sub-states of Management_TrainDataVal state

- Management_TRNN: Initialization_Manager executes the next step of procedure Start of Mission. This step consists in Train Running Number entry with value extracted from configuration file.

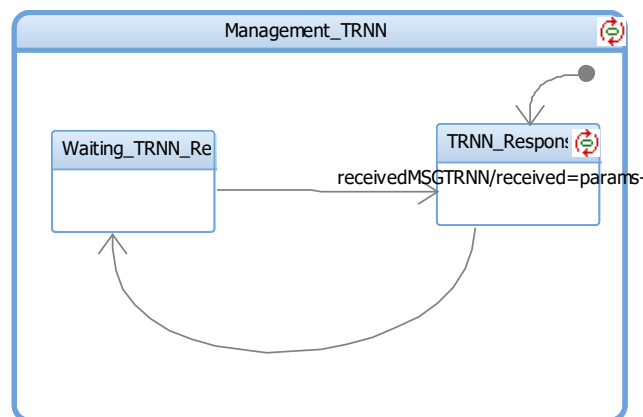


Figure 25: Sub-states of Management_TRNN state

- Waiting_start_enabled: Initialization_Manager executes the next step of procedure Start of Mission. This step consists in select the "Start" option enabled by ATC /ATC subsystem.

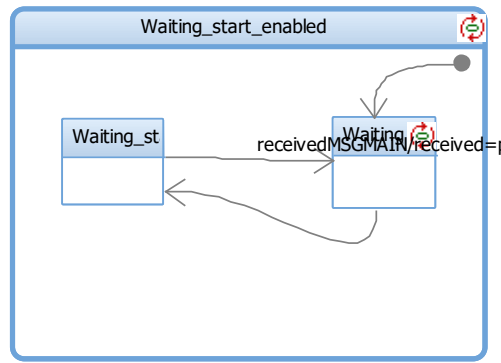


Figure 26: Sub-states of Waiting_start_enabled state

- **Ready:** Initialization_Manager executes the last step of procedure Start of Mission. This step consists in confirm the operating mode with which train will start its mission. Possible operating modes are those provided by ERTMS / ETCS standard.

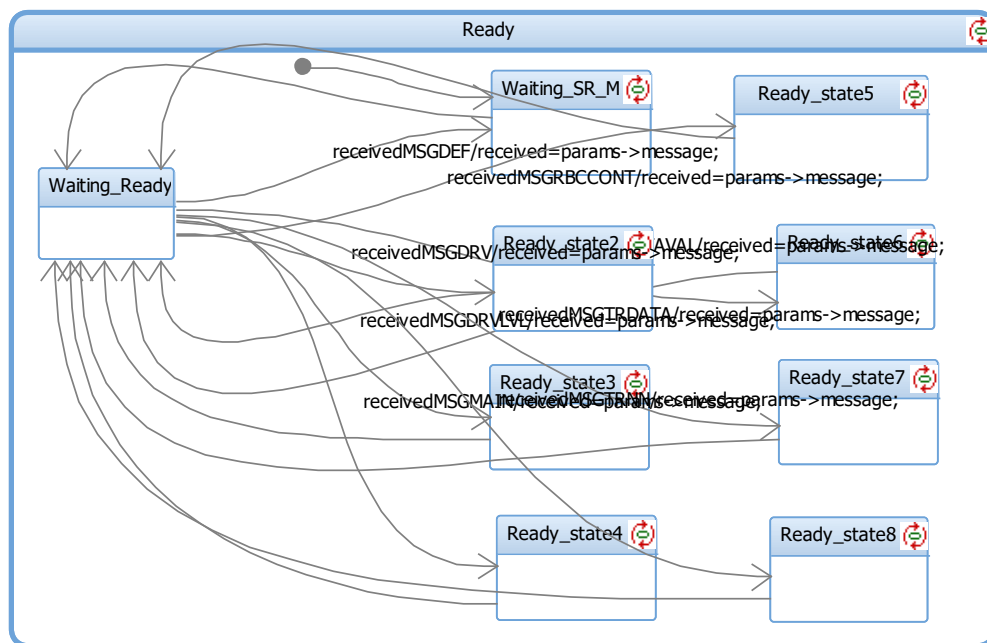


Figure 27: Sub-states of Ready state

- **Management Idle:** state reached by Initialization_Manager for faults or errors occurring during the procedure Start of Mission. Examples of possible faults or errors:
 - ATC doesn't receive from ATO the response message within 300 ms;
 - ATC doesn't receive from ATO the reply message due to loss of connection between ATC and ATO;
 - ATC receives from ATO a corrupt response message or with a sequence number different from expected number.

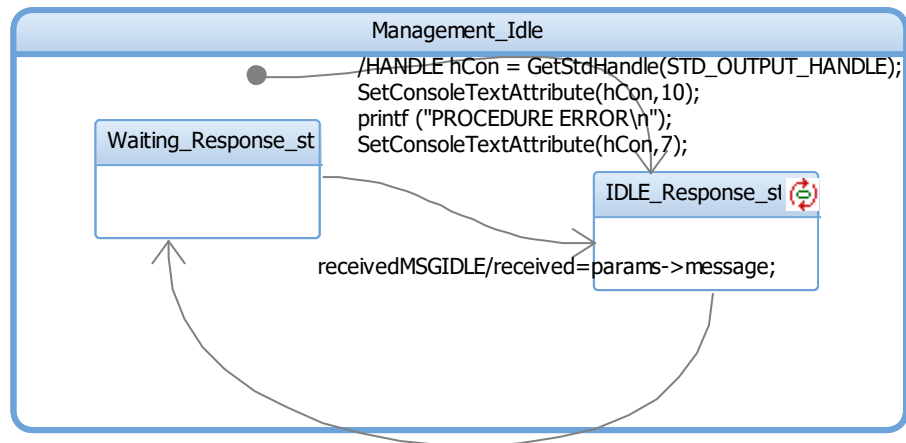


Figure 28: Sub-states of Management_Idle state

- Wait for Temination: state in which Initialization_Manager is waiting to go into sleeping mode or switch to Gestore_Marcia the control of ATO system.

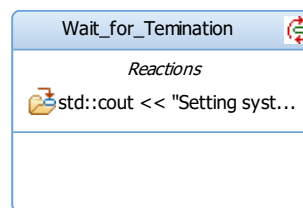


Figure 29: Wait_for_Termination state

- Sleeping: state in which all subsystems of ATO are in sleeping mode with the exception of ATS components.

If option is not enabled in the screen, Initialization_Manager creates an automatic reply message in which neither of options is selected.

4 Model Verification and Testing

The requirements that must be fulfilled during embedded software development are complex in comparison to standard software. Embedded systems interact with real-life environment and often involve large projects so the software is difficult to update once the product is deployed. Particularly in railway sector these systems are also called safety-critical systems where a failure can lead to catastrophic consequences (injured, loss of life and serious environmental damage). In terms of software development, increased complexity of products, shortened development cycles, and higher customer expectations of quality implicate the extreme importance of software testing.

Software Testing is an important component of software Quality Assurance, and today many software organizations are spending up to 40% of their resources on testing. This activity is very important for the software development, since testing is one of the last possibilities to evaluate the software product before its deployment to the final user.

Testing, validation, verification and also QA activities are especially important in the domain of embedded systems, such as aerospace or railway, due to the usually high dependability requirements (e.g., safety, reliability, and security). The aim of validation is to confirm that the developed product meets the user needs and requirements. Verification ensures that it is consistent, complete, and correct at the different steps of the life cycle. Testing is an activity with the aim to detect faults and can be used both for verification and for validation. A further important aspect is the application of QA for the certification of products, especially in safety-critical domains.

Software Testing can be considered one of the most expensive activities in the software development process and the optimization of resources is certainly one of the factors to consider. We need to consider some criteria regarding the testing coverage and test cases generation:

- Testing Coverage Criteria: it allows the identification of the percentage of the software that has been evaluated by a set of test cases
- Test Cases Generation Criteria: rules and guidelines used to generate a set of test cases "T" suitable to assess the quality of software. In functional testing this set is used to evaluate the adhesion of the software product to the requirements specification.

Today most organizations try to automate the testing activities to reduce the costs in support of these activities. But considering the complexity of all these activities, not all can be automated easily with the exception of the test execution. This assumes that the test

cases are already manually defined and written (or captured via a tool) and can be executed in an automated test execution environment in terms of scheduling, logging of results (success or failure), capturing of details of the failing environment, and so on. Automation of test case design (and hence test case creation) is another matter. In order to automate functional test case design, we need a formal description of the specifications of the software behaviour, resulting in a model of the software behaviour.

Into this context, Model-based Testing (MBT) appears to be a feasible approach to control the software quality. Indeed, MBT is based on the specification model that describes the behaviour of software system under test using software artifacts. This model allows to automatically generate test case and can be also used as the oracle to check whether the system behaves correctly. To apply this approach, some kind of formalism (such as mathematical notations provided by Formal methods, finite state machines, UML diagrams, etc.) and tools are required for design the specification model.

4.1 Adopted Methodology

In this work we have adopted a Model-Based Testing (MBT) approach to exploit the model of ATO system during the test authoring phase. We want to verify if the ATO system behaviour shown by the model is compliant with ATO System specification, defined during the analysis phase. Specifically, the verification has been focused on the functional behaviour compared to functional requirements defined in the ATO System Requirements Specification (SRS) [19] through the execution of functional testing. A testing process provides several steps that are summarized in the following workflow:



Figure 30: Testing Methodology Workflow

- **MODELLING:** This step involves the modelling of ATO System using IBM Rational Rhapsody tool. The result is the ATO system model which describe the system

behaviour through SySML constructs. The created model, which has been illustrated in detail in Figure 30, represents the software System Under Test (SUT).

- **TEST SUITE GENERATION:** For test case generation a first approach was to use the ATG tool provided by the Rhapsody suite [23]. In this way we can use the system model as input to ATG and obtain automatically test cases in output. However we haven't achieved good results. The ATG tool has proved insufficient to test the model exhaustively; in fact, some function blocks nested within states of the various statecharts have been not considered, thus leaving hidden some logical or functional errors. For this reason the instrument ATG is useful to generate test cases for simple models, but in the case of complex models is better to adopt other methodologies with the aim of achieving full coverage of the system (necessary condition in a safety-critical environment for railway transport). Our solution was been to build a "test environment", i.e. a model (with its own statechart) of the other components external to ATO, able to provide input and/or events needed to simulate the behaviour of ATO system components involved in a specified test case. Each test case verifies one or more specific functional requirements which can include several components of ATO system.
- **TEST EXECUTION:** Each test case was executed by the simulation of the realized ATO system model. The simulation is a method that allows to simulate an object or set of objects which form a model, in order to highlight a sequence of states activation within the realized statechart, through a series of inputs/events, sent by external entities involved in this context. IBM Rhapsody tool allows to understand what the system is doing and to detect its behaviour through the animation of the statecharts in execution. During test execution, the sequence of operations performed by the various components, and significant values were recorded in a special log file.
- **ANALYSIS RESULTS:** Verification can be difficult to accomplish because the system response must be determined and then compared to an expected response and/or verify that we are in the correct 'state' in our system. The degree that verification is used and the means for performing verification will vary widely with application type and test objectives.

In particular, work is focused on the generation and execution of test cases related to the **initialization of the train**.

4.2 Critical aspects of the model

From the analysis of the problem we have identified any critical aspects of the model due to the requirements from the performance of the procedure SoM and the communication protocol DMI-CPU32. Below are the critical issues identified that need to be tested:

- Correct answer to the message from ATC

- the block `Initialization_Manager` has successfully performed the SOM procedure if the ATC simulator sends to ATO the right message sequence.
- the block `Initialization_Manager` properly handles an error in case the order of received messages was uncorrected; the system goes into the state `Management_Idle` and the SOM procedure is aborted.
- ATO replies to every message from ATC within 300 ms.

First of all we have analyzed the critical issues due to the communication protocol and the proper response to incoming messages is the main issue that needs to be treated. It might seem a task that should be carried out by the protocol but in reality if the `Initialization_Manager` can not distinguish the type of message that has come and make the right answer, the communication can not be done properly. The other issue due to the communication protocol is the timing of responses. In fact communication DMI-CPU32 provides that if the ATC does not receive a response from DMI within 100ms try to postpone the message and if not receive further response considers the device inoperative and enter in IDLE state. So `Initialization_Manager` must be able to compose the replies within this time.

The last critical issues identified concerns the procedure Start of Mission: in fact this involves a certain sequence of messages and replies with the inclusion of data or even confirmation of already stored data. The sequence in this case does not have so many variations so can be considered static in the sense that can be used a single trace for testing the model.

4.3 Model Testing

In this work i have checked the Start of Mission (SoM) procedure, ensuring the proper execution of all required intermediate steps as reported in [8]. In this way the consistency of the system behaviour with respect to functional requirements `ATO_SYS_3030` to `ATO_SYS_3100` was verified. To test the SOM procedure we have modeled two blocks corresponding to external systems ATS and ATC. The first establishes the communication ATS-ATO and sends the *wake-up* message. The second one establishes the communication ATC-ATO and is responsible for sending messages related to different views that the virtual DMI must show during the SOM procedure. To accomplish this, the block ATC simulator reads the sequence of messages to be sent by a specific text file.

- Test to check that between the receipt of the message and sending the response the elapsed time is less than 100 ms
- Test to check that for every message received the correct answer was produced
- Tests to verify that the procedure was carried out in the correct order

In total we have defined some test cases that have allowed us to test the system in a comprehensive manner.

- Some test cases made use of traces detected by the real system
- Some test cases made use of modified tracks with incorrect sequence of messages
- Some test cases made use of modified tracks with incorrect messages

At the first execution, some tests have failed highlighting the presence of some errors in the model. All tests have detected that replies are sent to ATC with correct timing.

The error that compromise the correct execution of the tests is due to a problem of the composition of a particular type of message. The problem is related to an incorrect sequence of decoding / encoding of the message content due to misinterpretation of the bit sequence. Merely correcting the direction of reading the contents the problem was solved.

Following this the model was modified in order to eliminate anomalies and then tested again, running the tests defined above for the second time.

The results obtained after the second run have been significantly better than the first, in fact all tests were successful and we have reached 100% coverage in terms of states.

4.4 Integration Testing

The system ATO has been integrated with other on-board systems (ATC and interface of the train) and also put into communication with external systems (ATS). Tests were made to test the operation of the procedure Start of Mission with all real systems.

The main error that was detected refers to the sequence of messages to make it clear to ATC that has been pressed a button. The protocol DMI-CPU32 states that the pressure of a button is detected when there is a succession of messages with the value relative to the button that varies from low to high and back to low. In our model, this sequence of messages was sent too quickly: in practice the 3 necessary messages were sent one after the other. To repair the problem was necessary to slow down the sequence by sending multiple copies of the same message before changing the signal value. So the sequence from low-high-low becomes:

- 10 low
- 10 high
- 10 low

In this way ATC is able to detect the change of state of the button and to draw up in time the pressure of the button.

Solved this problem was possible to conclude the initialization procedure of ATO and then test the operation of the other components of the ATO itself. In the end it was possible to simulate the circulation of four trains simultaneously on a test track.

Part 2

1 Introduction to Social Media Monitoring

In this second part, will be presented the design and construction of a platform for crawling Social Network: in particular will be presented the use of the Twitter API and how they can be used to create a database of messages. In addition to the main process for the recovery of the messages we will be presented a number of secondary processes useful to the realization of a dashboard for displaying the results of the crawling and for the analysis of the messages retrieved.

1.1 Social Media Monitoring

The World Wide Web has become an active publishing system and is a rich source of information, thanks to contributions of hundreds of millions of Web users. The growths of online Social Networks in scale and amount of information are immense in recent years. Part of this public expression is carried out on social networking and social sharing sites (Twitter, Facebook, Youtube, etc.), part of it on independent Web sites powered by content management systems (CMSs, including blogs, wikis, news sites with comment systems, Web forums). Content published on this range of Web applications includes information that is newsworthy today or valuable to tomorrow's historians[24].

The analyses of the structure of online social networks have thus drawn much research interests[25]. Before the analyses, the information and the characteristics of the structure have to be obtained. However, the complexity of today's web technologies imposes challenges for collecting the data. The increasing popularity of online social networks (OSNs) has gathered hundreds of millions of users. OSNs have become a platform for people to easily communicate and share information, particularly with the sophisticate smartphones. Since the structures of OSNs will be able to reflect the reallife society in certain extend, the structure and the information shared in OSNs are of interests for different communities. For instance, sociologists regard OSNs as a venue for collecting relationship data and study online human behaviours. Marketers, in contrast, seek to exploit information about how messages spread so as to design viral marketing strategies. For network engineers,

understanding OSNs improves the design of interconnected systems so as to provide better user experience. In order to analyze the structure of an OSN, information regarding the network structure is needed.

The depth and quality of data that can be harvested from Social Media Monitoring tools has evolved significantly in the last years. In [26] has been evaluated the performance of some tools across the following general criteria: Query Set Up, Data Quality, Ease of Data Management, and User Interface & User Experience. Results of that comparison are the following:

- Alterian: Identify common themes among ostensibly disparate conversation
- Brandwatch: Hands on raw data management
- MutualMind: Threaded Facebook discussions
- Radian6: Sophisticated built-in engagement tool; robust dashboard
- Synthesio: Extremely flexible tool all around

What emerged were two distinct categories under which the tools might be classified:

- **Real Time Monitoring and Community Management:** tools that fall under this category seem best suited for monitoring and managing social media communities on a day-to-day basis. These tools are characterized by easily customizable dashboards to make the data a bit more digestible and by the ability to connect various social media accounts.
- **Research & Analysis:** these tools make the analysis of vast quantities of conversation data more manageable, and are generally better suited to monitoring social media performance over the long term. Access to historical data also makes these tools ideal for the development and evaluation of social media strategies.

Social media is especially important for research into computational social science that investigates questions using quantitative techniques (e.g., computational statistics, machine learning and complexity) and so-called big data for data mining and simulation modelling[27]. Social media scraping and analytics provides a rich source of academic research challenges for social scientists, computer scientists and funding bodies. Challenges include:

- **Scraping**—although social media data is accessible through APIs, due to the commercial value of the data, most of the major sources such as Facebook and Google are making it increasingly difficult for academics to obtain comprehensive access to their 'raw' data; very few social data sources provide affordable data offerings to academia and researchers.
- **Data cleansing**—cleaning unstructured textual data (e.g., normalizing text), especially high-frequency streamed real-time data, still presents numerous problems and research challenges.

- **Holistic data sources**—researchers are increasingly bringing together and combining novel data sources: social media data, real-time market & customer data and geospatial data for analysis.
- **Data protection**—once you have created a 'big data' resource, the data needs to be secured, ownership and IP issues resolved (i.e., storing scraped data is against most of the publishers' terms of service), and users provided with different levels of access; otherwise, users may attempt to 'suck' all the valuable data from the database.
- **Data analytics**—sophisticated analysis of social media data for opinion mining (e.g., sentiment analysis) still raises a myriad of challenges due to foreign languages, foreign words, slang, spelling errors and the natural evolving of language.
- **Analytics dashboards**—many social media platforms require users to write APIs to access feeds or program analytics models in a programming language, such as Java. While reasonable for computer scientists, these skills are typically beyond most (social science) researchers. Non-programming interfaces are required for giving what might be referred to as 'deep' access to 'raw' data, for example, configuring APIs, merging social media feeds, combining holistic sources and developing analytical models.
- **Data visualization**—visual representation of data whereby information that has been abstracted in some schematic form with the goal of communicating information clearly and effectively through graphical means. Given the magnitude of the data involved, visualization is becoming increasingly important.

The analysis of social networks has had a significant development, especially for commercial applications in marketing and advertising campaigns. In fact there are many tools that offer paid services for semantic and sentiment analysis to companies that want to evaluate the perception of its brand by users, the trend of a particular advertising campaign and all that can be extracted by the enormous mass data that social networks provide. The analysis tools can be divided into two categories: those that offer only the technology to retrieve information leaving the user the task of analyzing and those that offer a complete service.

In the first group we can insert **80legs** [28] that provides a web crawling directly from the website or through the API. 80legs gives the ability to customize crawl by providing a set of options that specify how crawl will run. These options are:

1. A list of URLs which tell 80legs where to start the crawl
2. A list of criteria to explain to 80legs what data to scrape from each crawled URL, as well as what URLs to crawl next
3. Other options to control the crawl, such as the number of total URLs to crawl.

Another tool that allows the retrieval of information from the web is **Visual Web Ripper** [29] that contains a wealth of advanced features that enables user to harvest data from even the

most difficult websites. Visual Web Ripper can be configured to download complete content structures, such as product catalogs. **HeliumScrapper** [30] can also be inserted in the first group because is a web scraping tool that can be trained to extract specific information from web sites. The results can be exported in a variety of formats. **PromptCloud** [31] is a classic web crawling model where is taken the list of sites that user would like crawled and do vertical-specific crawls. It's possible to provide PromptCloud with a list of keywords that gets fed into crawler. The crawler then continuously looks for matching tweets to that list of keywords as tweets gets published. All these tweets are later converted into a structured format with other associated information. In paper [32] authors propose one such tool called **Intention Insider** which has been developed at HP Labs in close collaboration with business units and a few selected customers. The tool can ingest content from online forums or from uploaded files and quickly sift through very large amounts of comments to extract intention information. This information is loaded into a data warehouse to be correlated with other structured data and queried to produce interactive reports and dynamic visualizations that facilitate its exploration at detailed and aggregate levels.

The second group of tools is much wider because the market requires a comprehensive toolset that directly provide the test results. These include **Openamplify** [33] that is an Natural Language Processing (NLP) analytic engine that processes text to extract valuable knowledge from social media conversations. This tool gives a picture of brand's health or campaign's performance. OpenAmplify analyzes any provided text, structured or unstructured, without the need for training or special vocabularies, returns a set of "signals", each of which describes a specific aspects of the text's meaning, sentiment, intent, style, or other characteristics and delivers the signals ranked and organized in useful ways. **Clarabridge Intelligence Platform** [34] gets a complete view of customers' experience. The Clarabridge Intelligence Platform harnesses all available sources of consumer feedback, including multiple survey types, contact center agent notes, social media, chat, voice, email, warranty notes, and much more. The Clarabridge Intelligence Platform's core functionality includes text analytics, context-sensitive sentiment analysis, linguistic categorization, and emotion detection. Clarabridge Social seamlessly gathers and accurately analyzes any online customer data, whether structured or unstructured, from any social media source and any online review site. Clarabridge Social connects to all popular sites including Facebook, Twitter, Trip Advisor, and Bookings.com, and integrates with social media management software such as Sysomos, Radian6, and many more for a comprehensive view. **Brandwatch Analytics** [35] is a web-based social media monitoring platform designed to allow users to get the most out of the social media data important to their business. It is focused on Customer Experience. Its main use cases are: Brand/reputation management, Finding influences/advocates, Market research, Campaign, Crisis management, Community management, PR, Customer Services, SEO and Lead generation. Channels feature allows to track any public Facebook page or Twitter account without the need for admin rights. **Opinion Crawl** [36] allows Companies and agencies to order in-depth reports monitoring online image of an entity - a company, a product, or an individual. The crawlers process large

amounts of various Web sources - blogs, news sites, forums. The reports are produced on an ongoing basis and emailed to the client. The reports are broken by day/week/month, and contain current and trend charts, key concepts associated with the topic, and references to source documents. Sentiment API allows client applications to assess sentiment on a Web page or a piece of text, e.g. a blog comment. **Social Report** [37] is a social network analytics solution that allows to track social network accounts just the same way it's possible to track the performance of websites. Social Report tracks and monitors social network accounts and gives user tools to manage marketing initiatives. Social Report offers powerful insights into social accounts: membership trends, activity and engagement, thoughts and feelings of members, their interests, their geographical distribution, education levels, gender, employment, and countless other metrics. **Mozenda** [38] is a web scraping service used by many well known brands. The Agent Builder supports the creation of agents that collect specific information from web sites. These are created in a Windows environment and submitted to the service where they are executed. The Web Console allows agents to be run and scheduled and export and publish the results of a search. **Beevolve** [39] monitors brand mentions, schedules and launch social media posts and measure resulting sales and engagement for those posts.

Meltwater's [40] online intelligence platform analyzes digital documents daily to extract precise, timely business insights that help executives understand their markets, engage their customers, and master the new social business environment. Meltwater PR solutions help public relations and marketing communications professionals build brands and drive growth by effectively engaging media influencers. Meltwater social media marketing solutions combine deep social media monitoring with efficient social engagement to help creating more effective marketing campaigns across large social communities. **Viralheat's** [41] sentiment analysis tool allows user to understand the sentiment of online mentions for business, brands, and products. Identifies the sentiment of social mentions across multiple social platforms and pulled detailed analysis of what users are saying about a product or service. This tool allows view sentiment of social posts in real-time and pull sentiment analytics from Facebook, Twitter, Instagram, and Tumblr. **SAS Sentiment Analysis** [42] automatically rates and classifies opinions expressed in electronic text. It collects text inputs from websites, social media outlets and internal file systems, and then puts them in a unified format to assess relevance to predefined topics. Reports identify trends or emotional changes, and an interactive workbench allows subject-matter experts to refine sentiment models. **Dataminr** [43] transforms real-time data from Twitter and other public sources into actionable signals, identifying the most relevant information in real-time for clients in Finance, the Public Sector, News, Security and Crisis Management. In partnership with Twitter, Dataminr developed and launched Dataminr for News, which alerts journalists to breaking news in advance of traditional sources and is now used by hundreds of news organizations globally. Most recently, Dataminr launched a product for security and crisis management watch centers that warns the world's largest corporations to emerging threats and crises, ensuring that a corporation's physical assets and employees are protected. **Tracx**

[44] is a company with a SaaS platform for sophisticated brand marketers who want to do more than monitor their social media presence, but actually manage it. The company provides an end-to-end solution that indexes the entire social web and delivers the most relevant, high impact audiences and conversations by capturing a 360 degree view of activity around a brand. **ROIALTY** [45] is the digital loyalty platform (web, social, mobile) that allows a brand to develop the potential of 'engagement' in social media & digital communities by increasing their involvement through Gamification & rewarding. It gives a boost to awareness, conversions and purchases on the e-commerce and retail channels monitoring the full range of interaction metrics needed to measure the ROI (Return On Investment) of each digital campaign. ROIALTY rewards authenticated users connecting their social profiles and offers them some targeted 'missions', based on their socio-demographics, preferences and interests. Each mission engages the user in content creation & publishing on blogs and social media or in promotion of product/service initiatives through likes and sharing as well as participating in surveys.

1.2 Twitter

Twitter is a social network that deals with free microblogging, devised by the American Jack Dorsey and developed by Obvious Corporation in San Francisco [46]. The service offered to members is the inclusion of messages, called 'tweets', consisting of a maximum of 140 characters.

Since its creation and networking in March 2006, Twitter has taken a prominent role within the set of social networks on the Web, reaching more than 250 million active users [47], that found in the service a quick way to interact with the rest of the world.

In addition to simple text, within the tweet you can enter keywords, called hashtags (preceded by a "#"), and links to other sites, usually abbreviated URL services via shorting. Messages posted by members are by default rendered visible to anyone, whether registered or not in the service, while you can make your tweets private so that they can only be read by authorized persons [48]. The inclusion of the messages is made possible not only through the social network site, but also by a number of external applications and, limited to a few countries, via SMS. The ability to send messages via different devices and applications, is one of the strengths of the social network.

Subscribers to the service have the opportunity to follow other registered users: they assume in this case the name of 'followers' and have the ability to view in their own "home page" messages posted by those users. It is also possible to follow lists of users, in other words lists created by other subscribers in which is included a variable number of users. Another important factor for the service is the ability to respond to messages from any other registered user, thus creating conversations online. A member can forward the message to another user who is following, so that it is visible to all their followers. This is called retweet and is reported in messages prefixing characters RT to the original text.

It should be noted that the conversations and personal status, calculated on a sample of tweets harvested from social networks, almost reach 90% of the total posts, while 37.55% of the total is made up of messages in response to other tweets. As for spam messages and self-promotion (ie tweet purposes only advertising placed by companies) they are limited to 9.6%. From these statistics it is possible to deduce how Twitter has become one of the most effective means to share experiences and how users can use it for communicating with each other, getting closer to the original idea of Jack Dorsey to create a service similar to SMS, but applied to groups of people and available on the web.

Since the beginning, the study of Twitter has proved of huge interest, being a social network popular, dynamic and where users, through their tweets, help to keep the public informed of what is happening around them. In 140 characters of a message are told life experiences both personal and about the world that surrounds users, while the ability to include links to other sites, as well as images and videos, are additional methods to disseminate what is most important in web.

2 Twitter Vigilance Analysis

The analysis of social networks has become the leading source of news, comments, opinions on any subject. As was pointed out in Chapter 1, there are already various tools, paid or not, which analyze and extract information from various social channels. Twitter Vigilance was designed as a platform for the search of messages on Twitter and for the analysis of such messages both from a numerical point of view (to highlight the daily peaks) and from a semantic point of view (for identify what refers peaks of tweets).

The idea of creating a platform for the monitoring of social networks is created within a collaboration between UNIFI DISIT lab, LAMMA and CNR IBINET. The main purpose is to investigate and to build specific and reliable metrics dashboard to monitor weather related Twitters. Since this study was the project of Twitter Vigilance Platform to provide a tool to study the content of the social network that was shared and adapted to different contexts, such as for monitoring city services, critical events and conditions, user behaviour, city response to events, etc.. Indeed, the tool that will be made will also be used in other projects related to smart cities.

Specifically, the main objective of the collaboration mentioned above is to analyze whether can be used the information flow of Twitter as a good social indicator of certain weather events, such as severe weather alerts or heat waves. Therefore in the design of Twitter Vigilance Platform has been introduced the concept of "thematic channel" in which research is collected concerning a certain topic, as can be weather alert or particular events as Expo.

A tool of this kind can not be separated from the management of users and the distinction of roles: basically will be split functionality between the User and the Administrator. A user must be able to create its channels and view only his own, but may be able to include in its channels every search included in the system even if inserted by another user. The administrator must be able to manage all channels of all users as well as monitor all system activity.

Since the platform is designed to be an analytics tool, a key part of the system is displaying the results of statistics and analyzes performed on messages within the channels. To achieve this purpose it was decided to create a dashboard that can show to the user, through graphs and charts, information that may be useful to explore its analysis.

In the next few paragraphs will be initially highlighted the main requirements of the platform and then highlighted the main issues to be addressed.

2.1 Requirements

This section contains the requirements of the platform Twitter Vigilance. The whole project is based on the concept of channel: this is the key idea that led to the design and creation of the Twitter Vigilance platform. The idea is to allow the user to create thematic channels in which enclose the research that the system must perform. Obviously the individual channel is associated with the user who creates it and then there could be more channels with the same name but associated with different users. This thing can not happen with the research: research must be unique within the system but can be associated with multiple channels. This difference comes from the fact that research with the Twitter API, if carried out with the same parameters, produce the same results. Below there are given the high-level requirements identified for the platform:

1. The system must issue queries to Twitter to retrieve all messages of interest.
2. The system must be based on the concept of thematic channel
 - a. To a channel must be associated one or more searches
3. The system must be able to follow explosives and slow events
4. The system must be multi-user
5. The system must provide a dashboard for viewing analysis results.

First requirement is perhaps the most important and the one that involves the major design difficulties: these are mainly due to the limitations and characteristics of the APIs provided by Twitter. In the next section we will describe the main Twitter API to access the message flow.

Second requirement is an innovation compared to the tools that can be found on the market of Social Media Monitoring. Indeed none of those who were tested can group search within a thematic channel: most are single theme or allow you to enter search without a specific grouping. This requirement does not directly involve retrieving messages from Twitter, since this is based on searches, but has a positive side in the analysis because statistics will be calculated on the entire body of messages that is contained in a channel as well as for each single search. In fact queries to Twitter are not based on the channel but on the search: why even if a search is connected to multiple channels, the system must perform a single query into Twitter.

The third requirement expresses the concept that the system has to adapt to the change in speed of the individual channels in relation to the occurrence of events that can experience an increased number of recoverable messages.

The fourth requirement enforces that the system has an authentication system allowing to distinguish between a user and another and between the various types of users.

The fifth requirement is more an interface requirement but is a fundamental requirement to characterize the platform on which the display of the analysis results is essential otherwise it

would not be a platform for the analysis of social networks but simply a platform for message retrieval.

2.2 Twitter API

As outlined above, the first target of the platform is to recover from Twitter all the messages that correspond to the Search that has been set in the system. In order to access messages, Twitter provides some API with different characteristics.

There are two types of APIs delivered by Twitter: REST API and Streaming API. Since version 1.1 of Twitter API is necessary to log in with OAuth for all requests, including search and streaming. Both API returns data in JSON format. Each API returns the same data, even when their JSON structures aren't. Regardless of the API used to search twitter messages, main effort is to built a valid query to Twitter databases. The query can have operators that modify its behaviour exactly like searches performed in Twitter website. The available operators are described in the following table:

Operator	Finds tweets...
<i>watching now</i>	containing both "watching" and "now". This is the default operator.
<i>"happy hour"</i>	containing the exact phrase "happy hour".
<i>love OR hate</i>	containing either "love" or "hate" (or both).
<i>beer -root</i>	containing "beer" but not "root".
<i>#haiku</i>	containing the hashtag "haiku".
<i>from:alexiskold</i>	sent from person "alexiskold".
<i>to:techcrunch</i>	sent to person "techcrunch".
<i>@mashable</i>	referencing person "mashable".
<i>superhero since:2015-07-19</i>	containing "superhero" and sent since date "2015-07-19" (year-month-day).
<i>ftw until:2015-07-19</i>	containing "ftw" and sent before the date "2015-07-19".
<i>movie -scary :)</i>	containing "movie", but not "scary", and with a positive attitude.
<i>flight :(</i>	containing "flight" and with a negative attitude.
<i>traffic ?</i>	containing "traffic" and asking a question.
<i>hilarious filter:links</i>	containing "hilarious" and linking to URL.
<i>news source:twitterfeed</i>	containing "news" and entered via TwitterFeed

Table 3: Operators for Twitter query

The REST APIs provide programmatic access to read and write Twitter data, author a new Tweet, read author profile and follower data, and more. The REST API identifies Twitter applications and users using OAuth; responses are available in JSON. Twitter Search APIs are part of Twitter's v1.1 REST API. It allows queries against the indices of recent or popular Tweets and behaves similarly to, but not exactly like the Search feature available in Twitter mobile or web clients, such as Twitter.com search. Search API allows a separate rate limited bucket of requests for each user. Rate limits are divided into 15 minutes intervals and all

endpoints require authentication. "GET search/tweets" is the API that returns a collection of relevant Tweets matching a specified query. The resource URL is: "<https://api.twitter.com/1.1/search/tweets.json>". Parameters that can be passed to "GET search/tweets" are:

- **q** (required): a UTF-8, URL-encoded search query of 500 characters maximum, including operators. Queries may additionally be limited by complexity. Example Values: *@noradio*
- **geocode** (optional): returns tweets by users located within a given radius of the given latitude/longitude. The location is preferentially taking from the Geotagging API, but will fall back to their Twitter profile. The parameter value is specified by "latitude,longitude,radius", where radius units must be specified as either "mi" (miles) or "km" (kilometers). Note that you cannot use the near operator via the API to geocode arbitrary locations; however you can use this geocode parameter to search near geocodes directly. A maximum of 1,000 distinct "sub-regions" will be considered when using the radius modifier. Example Values: *37.781157,-122.398720,1mi*
- **lang** (optional): restricts tweets to the given language, given by an ISO 639-1 code. Language detection is best-effort. Example Values: *eu*
- **locale** (optional): specify the language of the query you are sending (only *ja* is currently effective). This is intended for language-specific consumers and the default should work in the majority of cases. Example Values: *ja*
- **result_type** (optional): specifies what type of search results you would prefer to receive. The current default is "mixed." Valid values include:
 - mixed: Include both popular and real time results in the response.
 - recent: return only the most recent results in the response
 - popular: return only the most popular results in the response.

Example Values: *mixed, recent, popular*

- **count** (optional): the number of tweets to return per page, up to a maximum of 100. Defaults to 15. This was formerly the "rpp" parameter in the old Search API. Example Values: *100*
- **until** (optional): returns tweets created before the given date. Date should be formatted as YYYY-MM-DD. Keep in mind that the search index has a 7-day limit. In other words, NO tweets will be found for a date older than one week. Example Values: *2015-07-19*
- **since_id** (optional): returns results with an ID greater than (that is, more recent than) the specified ID. There are limits to the number of Tweets which can be accessed through the API. If the limit of Tweets has occurred since the *since_id*, the *since_id* will be forced to the oldest ID available. Example Values: *12345*
- **max_id** (optional): returns results with an ID less than (that is, older than) or equal to the specified ID. Example Values: *54321*

- **include_entities** (optional): the entities node will be disincluded when set to false. Example Values: *false*
- **callback** (optional): if supplied, the response will use the JSONP format with a callback of the given name. The usefulness of this parameter is somewhat diminished by the requirement of authentication for requests to this endpoint. Example Values: *processTweets*

Example Request:

```
"GET https://api.twitter.com/1.1/search/tweets.json?q=%23freebandnames
```

```
&since_id=24012619984051000&max_id=250126199840518145&result_type=mixed&count=4"
```

In the following figure is presented an example result of query on Twitter using Search API.

```
{
  "statuses": [
    {
      "coordinates": null,
      "favorited": false,
      "truncated": false,
      "created_at": "Mon Sep 24 03:35:21 +0000 2012",
      "id_str": "250075927172759552",
      "entities": {
        "urls": [
          ],
        ],
      "hashtags": [
        {
          "text": "freebandnames",
          "indices": [
            20,
            34
          ]
        }
      ],
      "user_mentions": [
        ]
      },
      "in_reply_to_user_id_str": null,
      "contributors": null,
      "text": "Aggressive Ponytail #freebandnames",
      "metadata": {
        "iso_language_code": "en",
        "result_type": "recent"
      },
      "retweet_count": 0,
      "in_reply_to_status_id_str": null,
      "id": 250075927172759552,
      "geo": null,
      "retweeted": false,
      "in_reply_to_user_id": null,
    }
  ]
}
```

```

"place": null,
"user": {
  "profile_sidebar_fill_color": "DDEEF6",
  "profile_sidebar_border_color": "CODEED",
  "profile_background_tile": false,
  "name": "Sean Cummings",
  "profile_image_url": "http://a0.twimg.com/profile_images/2359746665/1v6zfgqo8g0d3mk7ii5s_normal.jp
eg",
  "created_at": "Mon Apr 26 06:01:55 +0000 2010",
  "location": "LA, CA",
  "follow_request_sent": null,
  "profile_link_color": "0084B4",
  "is_translator": false,
  "id_str": "137238150",
  "entities": {
    "url": {
      "urls": [
        {
          "expanded_url": null,
          "url": "",
          "indices": [
            0,
            0
          ]
        }
      ]
    },
    "description": {
      "urls": [

    ]
    }
  },
  "default_profile": true,
  "contributors_enabled": false,
  "favourites_count": 0,
  "url": null,
  "profile_image_url_https": "https://si0.twimg.com/profile_images/2359746665/1v6zfgqo8g0d3mk7ii5s_no
rmal.jpeg",
  "utc_offset": -28800,
  "id": 137238150,
  "profile_use_background_image": true,
  "listed_count": 2,
  "profile_text_color": "333333",
  "lang": "en",
  "followers_count": 70,
  "protected": false,
  "notifications": null,
  "profile_background_image_url_https": "https://si0.twimg.com/images/themes/theme1/bg.png",
  "profile_background_color": "CODEED",
  "verified": false,
  "geo_enabled": true,
  "time_zone": "Pacific Time (US & Canada)",
  "description": "Born 330 Live 310",
  "default_profile_image": false,
  "profile_background_image_url": "http://a0.twimg.com/images/themes/theme1/bg.png",
  "statuses_count": 579,

```



```

    "friends_count": 110,
    "following": null,
    "show_all_inline_media": false,
    "screen_name": "sean_cummings"
  },
  "in_reply_to_screen_name": null,
  "source": "<a>Twitter for Mac</a>",
  "in_reply_to_status_id": null
}
],
"search_metadata": {
  "max_id": 250126199840518145,
  "since_id": 24012619984051000,
  "refresh_url": "?since_id=250126199840518145&q=%23freebandnames&result_type=mixed&include_entities=1",
  "next_results": "?max_id=249279667666817023&q=%23freebandnames&count=4&include_entities=1&result_type=mixed",
  "count": 4,
  "completed_in": 0.035,
  "since_id_str": "24012619984051000",
  "query": "%23freebandnames",
  "max_id_str": "250126199840518145"
}
}

```

Figure 31: JSON example

There are four main “objects” that is possible to encounter in the API results: Tweets, Users, Entities and Places. Main fields that can be included in JSON response are described in the following table.

Field	Type	Description
annotations	Object	<i>Unused.</i> Future/beta home for status annotations.
contributors	Collection of Contributors	<i>Nullable.</i> An collection of brief user objects (usually only one) indicating users who contributed to the authorship of the tweet, on behalf of the official tweet author. Discussion. Example:

Field	Type	Description
		<pre>"contributors": [{ "id":819797, "id_str":"819797", "screen_name":"episod" }]</pre>
coordinates	Coordinates	<p><i>Nullable</i>. Represents the geographic location of this Tweet as reported by the user or client application. The inner coordinates array is formatted as geoJSON(longitude first, then latitude).</p> <p>Example:</p> <pre>"coordinates": { "coordinates": [-75.14310264, 40.05701649], "type":"Point" }</pre>
created_at	String	<p>UTC time when this Tweet was created.</p> <p>Example:</p> <pre>"created_at":"Wed Aug 27 13:08:45 +0000 2008"</pre>
current_user_retweet	Object	<p><i>Perspectival</i>. Only surfaces on methods supporting the <code>include_my_retweetparameter</code>, when set to true. Details the Tweet ID of the user's own retweet (if existent) of this Tweet.</p> <p>Example:</p>

Field	Type	Description
		<pre>"current_user_retweet": { "id": 26815871309, "id_str": "26815871309" }</pre>
entities	Entities	<p>Entities which have been parsed out of the text of the Tweet. Additionally see Entities in Twitter Objects. Example:</p> <pre>"entities": { "hashtags": [], "urls": [], "user_mentions": [] }</pre>
favorite_count	Integer	<p><i>Nullable</i>. Indicates approximately how many times this Tweet has been “favorited” by Twitter users.</p> <p>Example:</p> <pre>"favorite_count": 1138</pre>
favorited	Boolean	<p><i>Nullable. Perspectival</i>. Indicates whether this Tweet has been favorited by the authenticating user. Example:</p> <pre>"favorited": true</pre>
filter_level	String	<p>Indicates the maximum value of the filter_level parameter which may be used and still stream this Tweet. So a value of medium will be streamed on none, low, and medium streams. Example:</p> <pre>"filter_level": "medium"</pre>
geo	Object	<p>Deprecated. Nullable. Use the “coordinates” field instead. Discussion</p>

Field	Type	Description
id	Int64	The integer representation of the unique identifier for this Tweet. This number is greater than 53 bits and some programming languages may have difficulty/silent defects in interpreting it. Using a signed 64 bit integer for storing this identifier is safe. Use <code>id_str</code> for fetching the identifier to stay on the safe side. Example: <pre>"id":114749583439036416</pre>
id_str	String	The string representation of the unique identifier for this Tweet. Implementations should use this rather than the large integer in <code>id</code> . Discussion. Example: <pre>"id_str": "114749583439036416"</pre>
in_reply_to_screen_name	String	<i>Nullable</i> . If the represented Tweet is a reply, this field will contain the screen name of the original Tweet's author. Example: <pre>"in_reply_to_screen_name": "twitterapi"</pre>
in_reply_to_status_id	Int64	<i>Nullable</i> . If the represented Tweet is a reply, this field will contain the integer representation of the original Tweet's ID. Example: <pre>"in_reply_to_status_id":114749583439036416</pre>
in_reply_to_status_id_str	String	<i>Nullable</i> . If the represented Tweet is a reply, this field will contain the string representation of the original Tweet's ID. Example: <pre>"in_reply_to_status_id_str": "114749583439036416"</pre>
in_reply_to_user_id	Int64	<i>Nullable</i> . If the represented Tweet is a reply,

Field	Type	Description
		<p>this field will contain the integer representation of the original Tweet's author ID. This will not necessarily always be the user directly mentioned in the Tweet.</p> <p>Example:</p> <pre>"in_reply_to_user_id":819797</pre>
in_reply_to_user_id_str	String	<p><i>Nullable.</i> If the represented Tweet is a reply, this field will contain the string representation of the original Tweet's author ID. This will not necessarily always be the user directly mentioned in the Tweet.</p> <p>Example:</p> <pre>"in_reply_to_user_id_str":"819797"</pre>
lang	String	<p><i>Nullable.</i> When present, indicates a BCP 47 language identifier corresponding to the machine-detected language of the Tweet text, or "und" if NO language could be detected. Example:</p> <pre>"lang": "en"</pre>
place	Places	<p><i>Nullable.</i> When present, indicates that the tweet is associated (but not necessarily originating from) a Place. Example:</p>

Field	Type	Description
		<pre> "place": { "attributes": {}, "bounding_box": { "coordinates": [[[-77.119759 ,38.791645], [-76.909393 ,38.791645], [-76.909393 ,38.995548], [-77.119759 ,38.995548]]], "type": "Polygon" }, "country": "United States", "country_code": "US", "full_name": "Washington, DC", "id": "01fbe706f872cb32", "name": "Washington", "place_type": "city", "url": "http://api.twitter.com/1/geo/id/01fbe706f872cb32.json" } </pre>
possibly_sensitive	Boolean	<p><i>Nullable</i>. This field only surfaces when a tweet contains a link. The meaning of the field doesn't pertain to the tweet content itself, but instead it is an indicator that the URL contained in the tweet may contain content or media identified as sensitive content. Example:</p> <pre> "possibly_sensitive": true </pre>
quoted_status_id	Int64	This field only surfaces when the Tweet is a quote Tweet. This field contains the integer

Field	Type	Description
		value Tweet ID of the quoted Tweet. Example: <pre>"quoted_status_id": 114749583439036416</pre>
quoted_status_id_str	String	This field only surfaces when the Tweet is a quote Tweet. This is the string representation Tweet ID of the quoted Tweet. Example: <pre>"quoted_status_id_s tr": "11474958343903 6416"</pre>
quoted_status	Tweet	This field only surfaces when the Tweet is a quote Tweet. This attribute contains the Tweet object of the original Tweet that was quoted.
scopes	Object	A set of key-value pairs indicating the intended contextual delivery of the containing Tweet. Currently used by Twitter's Promoted Products. Example: <pre>"scopes": { "followers": false }</pre>
retweet_count	Int	Number of times this Tweet has been retweeted. This field is NO longer capped at 99 and will not turn into a String for "100+". Example: <pre>"retweet_count": 158 5</pre>
retweeted	Boolean	<i>Perspectival</i> . Indicates whether this Tweet has been retweeted by the authenticating user. Example: <pre>"retweeted": false</pre>
retweeted_status	Tweet	Users can amplify the broadcast of tweets authored by other users by retweeting. Retweets can be distinguished from typical Tweets by the existence of

Field	Type	Description
		<p>a <code>retweeted_status</code> attribute. This attribute contains a representation of the <i>original</i> Tweet that was retweeted. Note that retweets of retweets do not show representations of the intermediary retweet, but only the original tweet. (Users can also unretweet a retweet they created by deleting their retweet.)</p>
source	String	<p>Utility used to post the Tweet, as an HTML-formatted string. Tweets from the Twitter website have a source value of <code>web</code>. Example:</p> <pre data-bbox="858 853 1125 1173">"source": "\u003Ca href=\"http://itunes.apple.com/us/app/twitter/id409789998?mt=12\" rel=\"nofollow\"\u003ETwitter for Mac\u003C/a\u003E"</pre>
text	String	<p>The actual UTF-8 text of the status update. See <code>twitter-text</code> for details on what is currently considered valid characters. Example:</p> <pre data-bbox="858 1368 1125 1621">"text": "Tweet Button, Follow Button, and Web Intents javascript now support SSL http://t.co/9fbA0oYy ^TS"</pre>
truncated	Boolean	<p>Indicates whether the value of the <code>text</code> parameter was truncated, for example, as a result of a retweet exceeding the 140 character Tweet length. Truncated text will end in ellipsis, like this <code>...</code> Since Twitter now rejects long Tweets vs truncating them, the large majority of Tweets will have this set to <code>false</code>. Note that while native retweets may have</p>

Field	Type	Description
		<p>their toplevel text property shortened, the original text will be available under the retweeted_status object and the truncatedparameter will be set to the value of the original status (in most cases, false). Example:</p> <pre>"truncated":true</pre>
user	Users	<p>The user who posted this Tweet. Perspectival attributes embedded within this object are unreliable.</p>
withheld_copyright	Boolean	<p>When present and set to "true", it indicates that this piece of content has been withheld due to a DMCA complaint. Example:</p> <pre>"withheld_copyright": true</pre>
withheld_in_countries	Array of String	<p>When present, indicates a list of uppercase two-letter country codes this content is withheld from. Twitter supports the following non-country values for this field:</p> <p>"XX" - Content is withheld in all countries</p> <p>"XY" - Content is withheld due to a DMCA request. Example:</p> <pre>"withheld_in_countries": ["GR", "HK", "MY"]</pre>
withheld_scope	String	<p>When present, indicates whether the content being withheld is the "status" or a "user." Example:</p> <pre>"withheld_scope": "status"</pre>

Table 4: JSON fields

The Streaming APIs give developers low latency access to Twitter's global stream of Tweet data. Connecting to the streaming API requires keeping a persistent HTTP connection open. Twitter offers several streaming endpoints, each customized to certain use cases:

- **Public streams:** streams of the public data flowing through Twitter. Suitable for following specific users or topics, and data mining.
- **User streams:** single-user streams, containing roughly all of the data corresponding with a single user's view of Twitter.
- **Site streams:** the multi-user version of user streams. Site streams are intended for servers which must connect to Twitter on behalf of many users.

Twitter only allows to make a single streaming API OAuth connection for each twitter account that the app owns. An application can use a single connection to the streaming API, although users who access to application have separate accounts. "GET statuses/sample" API returns a small random sample of all public statuses. The Tweets returned by the default access level are the same, so if two different clients connect to this endpoint, they will see the same Tweets. Resource URL is: "<https://stream.twitter.com/1.1/statuses/sample.json>".

2.3 Main challenges

As highlighted by the platform requirements, the most critical are given by retrieving messages from Twitter. These are due to some limitations of the API that have led to very specific design choices to circumvent these limitations and have an efficient process of retrieving messages.

The first obvious limitation that involves both types of APIs is given by the corpus of messages that Twitter makes available for retrieval through the APIs. The Search API is not complete index of all tweets, but instead an index of recent Tweets. At the moment That index includes between 6-9 days of Tweets. Total rate limits of Streaming API is not documented. The documentation say up to 1% of the full firehose of tweets. So none of the two types of APIs allows you to recover the entire corpus of messages that is present on Twitter.

REST API v1.1's rate limiting model allows for a wider ranger of requests through per-method request limits. There are two initial buckets available for GET requests: 15 calls every 15 minutes, and 180 calls every 15 minutes. Search will be rate limited at 180 queries per 15 minute. Streaming API doesn't have these limitations because once applications establish a connection to a streaming endpoint, they are delivered a feed of Tweets, without needing to worry about polling or REST API rate limits.

Another limitation is the need for authentication that have the APIs: There are 2 types of authentication per-user and application-only. Rate limiting in version 1.1 of the API is primarily considered on a per-user basis — or more accurately described, per access token in

your control. If a method allows for 15 requests per rate limit window, then it allows you to make 15 requests per window per leveraged access token. When using application-only authentication, rate limits are determined globally for the entire application. If a method allows for 15 requests per rate limit window, then it allows you to make 15 requests per window — on behalf of your application. This limit is considered completely separately from per-user limits. Search API provides both authentication per-user and application-only: the per-user authentication can have multiple accounts associated with an application, each of which has a limit of 180 requests / 15-minute window, the application-only authentication allows for one account per application with a limit of 450 requests / 15-min window. Streaming API lacks these types of account, but has a limitation on the number of connections per account. Each account may create only one standing connection to the public endpoints, and connecting to a public stream more than once with the same account credentials will cause the oldest connection to be disconnected.

For what concerns total flow of tweets, the Search API returns up to 100 tweets per search and allows 720 requests per hour, giving a max of 72,000 tweets per hour. If each user who logs into an app asks to make search requests, then is possible to get up to 72,000 tweets per hour for every user. Streaming API has maxed out at around 3,000 tweets a minute, this delivers a maximum flow of 180,000 tweets an hour.

An important goal is to create a system that can adapt to the number of posts made available by twitter and that is able to retrieve them all. This is based on the speed of the channel is identified by the number of posts "new" retrieved via the API. In the case of the Search API if each request are recovered 100 new posts, it means that at that time there are many posts available and that probably not everyone is recovered but there may be losses. The objective must therefore be to reduce the interval between a request and the next in order to minimize the losses (saturations).

A final critical point is due to the dynamism of social networks: in fact there is some information that relates to the individual message depending on how many users interact with the message itself. When the message is retrieved, this brings the information representing the state of the message at that moment. This information, however, may change thereafter and must be retrieved in order to update the status of the message to the latest version.

3 Twitter Vigilance Design

To achieve the overall objectives set out in paragraph 2.1, has been designed a platform to address the concerns set out in the previous chapter efficiently and to provide analysis tools clear and self explanatory.

The platform is divided into 2 parts: a backend for retrieving messages and a frontend for setting up the channels by the user and for displaying the results. In this thesis will be presented both backend and frontend: in backend are present a process for recovering of the tweets and others processes for the calculation of graphics and performance of the system, frontend is an implementation of a dashboard to present some statistics on downloaded messages by means graphs and tables.

In the following paragraphs are presented the projects of the processes that are part of the backend of the platform and the design of the dashboard.

3.1 Backend

For the design of the backend of Twitter Vigilance platform were taken into account all the aspects listed above that restrict the use of the Twitter API. In particular, it was chosen to use the Search API as they offer a greater chance of filtering query results, given the number of parameters that provide. As for limiting the number of messages downloaded, which might suggest the choice of the Streaming API, it is avoidable by increasing the number of accounts associated with the application. With multiple accounts it's possible also to increase the number of requests that can be made. To manage a multi account, has been designed an architecture in which there is a central process that schedules the execution of individual search and a number of independent processes equal to the number of accounts that deal with perform searches in Twitter and store the results in the database. In this version of the platform have been used 5 accounts. In order to update the data of retrieved messages have created a separate process that once a day requires to Twitter data of messages stored in the database.

In the next few paragraphs will be presented requirements and architecture of the processes which are part of the backend.

3.1.1 **Requirements**

1. The system must use the Twitter Search API for researching Twitter based on the keywords set by users.
2. The system must store the results in a table in a relational DB.
3. The system should be divided into two subsystems:
 - a. a central process
 - b. various processes independent from each other and from the central process
4. The core process must start the processes to perform searches on Twitter
5. The core process must assign a ranking and a deadline to the various investigations based on the number of new messages downloaded.
 - a. according to the average number of new messages downloaded in the last two executions is assigned a ranking
 - i. based on ranking is calculated the new deadline of the search.
6. Each process needs to use a different account to the Twitter Search API.
7. Each process queries the database to take charge of the first search key whose deadline has expired.
8. Each process has to build the application to be submitted to Twitter using the Search API.
9. Every process must store the results obtained from the Search in the database.
10. The independent processes must access the DB exclusively so that the same search key is not taken over by two processes simultaneously.
11. The core process must monitor the number of requests made by each independent process, not to exceed the limit of 180 requests every 15 minutes.
 - a. If a process reaches retirement requests, pauses.
 - b. When the limit is reset, the process is reactivated.
 - c. The main process must make a reduction of the research by selecting a subset that is minimal: research whose results are contained in the other are not taken into account, this reduces the number of requests to Twitter without any loss of data.
12. The system must include a sub-process that update messages for the recovery of the number of "retweet" and "favorite".
13. The system must include a sub-process to retrieve "father" tweets.
14. The system must include sub-processes to compute data for graphics and tables.
 - a. Number of tweets for each Channel
 - b. Number of tweets for each Search
 - c. Number of retweets for each Channel
 - d. Number of retweets for each Search
 - e. Number of users for each Channel
 - f. Number of users for each Search
15. The system must store messages distinguishing between those direct and retweets.
16. The new research is created with the status set to "disactive".
17. A search must be activated only when it is associated with at least one active channel, otherwise the search is "disactive".

3.1.2 Backend Architecture

The main components of this architecture are the Scheduler and the Crawler.

The Scheduler is the core process of the entire system and it is a single process that is responsible for initiating the Crawler, updating the rankings associated with research based on the number of new tweets in the database in the last two versions of search and update the deadline of performing searches based on ranking assigned previously. The Crawler is responsible for the process to take charge of the research, to forward the requests to Twitter with the search parameters taking charge and store the downloaded messages in the database. Crawler can be instantiated multiple processes and each must have a Twitter account. Both processes read and write to the database, and conflicts are avoided by locking mechanisms of the tables and a mutex.

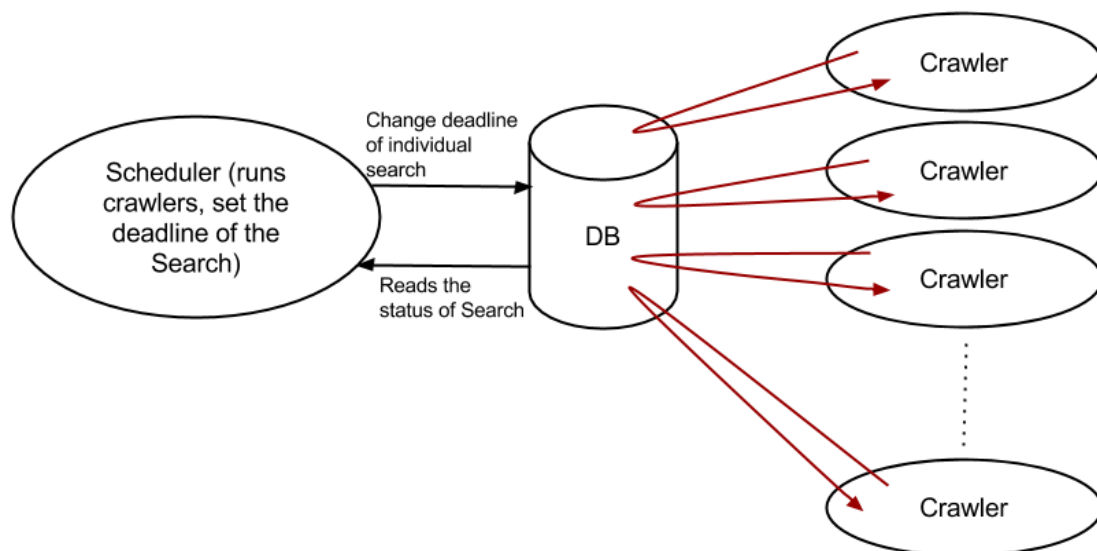


Figure 32: Crawling system architecture

3.1.3 DB structure

The Twitter Vigilance platform uses 23 tables: `account_twitter`, `adjectives_on_channel`, `ax_request_twitter`, `ax_twitter`, `ax_twitter_geocode`, `channel`, `chart_eventi_canali`, `chart_twitter`, `chart_twitter_canali`, `chart_twitter_retweet`, `chart_twitter_retweet_canali`, `chart_user`, `chart_user_canali`, `hashtags`, `hashtags_on_channel`, `keywords_on_channel`, `mentions`, `mentions_on_channel`, `process_list`, `retweet_count`, `users`, `variable`, `verbs_on_channel`. Those used in the process of retrieving messages from Twitter are just 7: `account_twitter`, `ax_request_twitter`, `ax_twitter`, `hashtags`, `mentions`, `process_list`, `variable`. The others are used for data analysis.

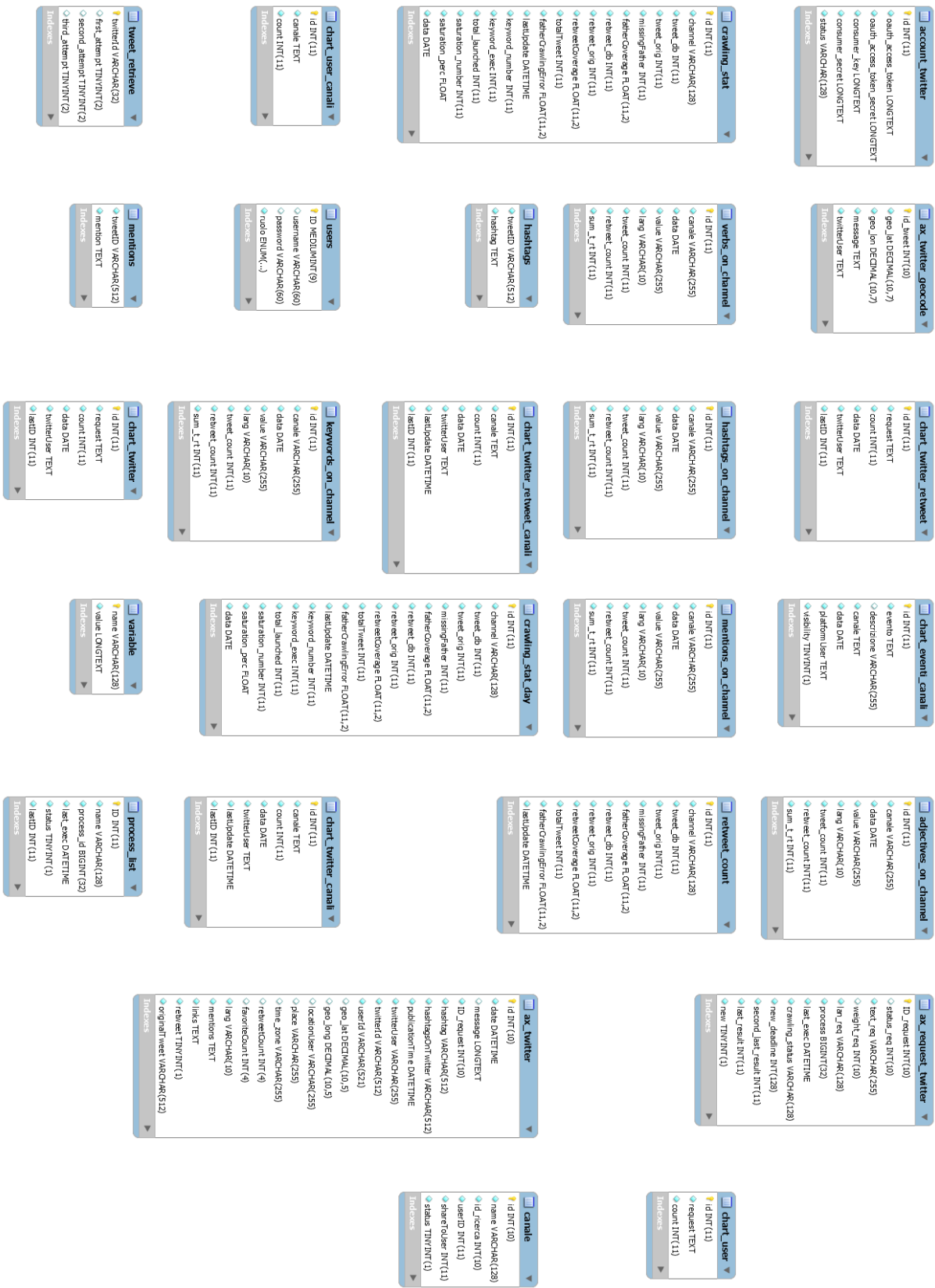


Figure 33: Database architecture

account_twitter

The table contains the keys to be used to connect with the Twitter API. The "status" field indicates whether the account is used by a process, or if it is available.

Column	Type	Null	Default	Description
<i>id</i>	int(11)	NO		
oauth_access_token	longtext	NO		Credentials for access to Twitter
oauth_access_token_secret	longtext	NO		Credentials for access to Twitter
consumer_key	longtext	NO		Credentials for access to Twitter
consumer_secret	longtext	NO		Credentials for access to Twitter
status	varchar(128)	NO	available	It indicates whether the account is already in use by some process

Table 5: account_twitter table

ax_request_twitter

The table "ax_request twitter" contains parameters of each request to the Twitter API. Also there are the fields necessary to schedule requests to Twitter (process, last_exec, crawling_status, new_deadline, second_last_result, last_result)

Column	Type	Null	Default	Description
<i>ID_request</i>	int(10)	NO		Identifier of requests
status_req	int(10)	YES	NULL	Indicates whether the application is active (1) or not (0)
text_req	varchar(255)	NO		The text of the research including the symbols
weight_req	int(10)	YES	NULL	Numeric indication of the priority of the request
lan_req	varchar(128)	NO		Language to be used as a parameter in the query
process	bigint(32)	NO		ID of the process that handles the request
last_exec	datetime	NO		Date and time of the execution of the request
crawling_status	varchar(128)	NO	completed	Call status (waiting, processing, completed)
new_deadline	int(128)	NO	0	Timestamp indicating when will the next run
second_last_result	int(11)	NO		Number of tweets inserted in the second last run

last_result	int(11)	NO		Number of tweets inserted in the last run
-------------	---------	----	--	---

Table 6: ax_request_twitter table

ax_twitter

The table "ax_twitter" contains the data of each tweet into the system. The tweets are unique thanks to the unique constraint on the field "twitterId".

Column	Type	Null	Default	Description
id	int(10)	NO		Index will be automatically incremented and assigned to each tweet inserted
date	datetime	NO		Date and Time when the tweet was inserted or updated in the table
message	longtext	YES	NULL	The message body in the tweet
ID_request	int(10)	NO		ID of the request as indicated in the table "ax_request twitter"
hashtag	varchar(512)	NO		Hashtags text set by administrator
hashtagsOnTwitter	varchar(512)	NO		List of hashtags contained in every tweet
publicationTime	datetime	NO		Date and Time when the tweet was published
twitterUser	varchar(255)	NO		Name of the user who sent the tweet
twitterId	varchar(512)	NO		ID of the tweet
userId	varchar(521)	NO		ID of user
geo_lat	decimal(10,5)	YES	NULL	Geographic location of tweet, latitude
geo_long	decimal(10,5)	YES	NULL	Geographic location of tweet, longitude
locationUser	varchar(255)	YES	NULL	Indicating the country of origin of User
place	varchar(255)	YES	NULL	Place associated with the content of the tweet
time_zone	varchar(255)	YES	NULL	Time zone associated with the tweet
retweetCount	int(4)	YES	NULL	It indicates the number of times that the tweet was forwarded

favoriteCount	int(4)	YES	NULL	It indicates how many times the tweet has been marked as favorite
lang	varchar(10)	NO		Indicates the language of message
mentions	text	NO		It indicates users mentioned in the message
links	text	NO		Indicates the link contained in the message both in the shortened version and in the extended
retweet	tinyint(1)	NO		Indicating whether the message is a retweet or NO
originalTweet	varchar(512)	NO		If the message is a retweet, indicates the ID of the original message

Table 7: ax_twitter page

ax_twitter_geocode

The table "ax twitter geocode" shows the geolocation of each tweet.

Column	Type	Null	Default	Description
<i>id_tweet</i>	int(10)	NO		Index associated with the index ID of table ax_twitter
geo_lat	decimal(10,7)	NO		Geographic location of this Tweet obtained from an external service of geolocation based on the "place" field of table ax_twitter, latitude
geo_lon	decimal(10,7)	NO		Geographic location of this Tweet obtained from an external service of geolocation based on "place" field of table ax_twitter, longitude
message	text	NO		The message body of the tweet
twitterUser	text	NO		Name of the user who sent the tweet

Table 8: ax_twitter_geocode table

canale

The table "canale" contains the associations between channels and searches. Each line represents the association between a channel, a research and the user that created the channel. This structure enables to maintain the uniqueness of the channels associated with a user. This does not prevent to have channels with the same name and same searches but linked to different users. There are also a field to share the channel with a user other than the owner and a field to define the status of the channel.

Column	Type	Null	Default	Description
id	int(10)	NO		ID of the association Channel-Search
name	varchar(128)	NO		Indicates the Channel name
id_ricerca	int(10)	NO		Indicates the identifier of the Search associated with the Channel (refers to field ID_request of the table "ax_request_twitter")
userID	int(11)	NO		ID of the user who added the channel (refer to ID field of the table "users")
shareToUser	int(11)	NO		id of user to share channel (typically a guest user)
status	tinyint(1)	NO	1	Channel status (active, disactive)

Table 9: canale table

chart_eventi_canali

It contains the labels that a user can apply to graphics to highlight a particular event.

Column	Type	Null	Default	Description
id	int(11)	NO		
evento	text	NO		event text to be displayed
descrizione	varchar(255)	YES	NULL	description of the event
canale	text	NO		channel to which the event is associated
data	date	NO		the event date
platformUser	text	NO		user who added the event
visibility	tinyint(1)	NO		visibility of the event (public or private)

Table 10: chart_eventi_canali table

chart_twitter

Tables "chart_twitter", "chart_twitter_canali", "chart_twitter_retweet" and "chart_twitter_retweet_canali" are used to store data necessary to render charts. In this way the updating of data can take place in the background with a separate process in such a way as not to affect the display speed of the graphs.

Column	Type	Null	Default	Description
<i>id</i>	int(11)	NO		the row identifier (auto_increment)
request	text	NO		search text
count	int(11)	NO		number of messages relevant to search on a certain day
data	date	NO		date to which refers the count of messages
twitterUser	text	NO		number of distinct users that have posted the messages

Table 11: chart_twitter table

chart_twitter_canali

Column	Type	Null	Default	Description
<i>id</i>	int(11)	NO		the row identifier (auto_increment)
canale	text	NO		channel name
count	int(11)	NO		number of messages related to the channel on a certain day
data	date	NO		date to which refers the count of messages
twitterUser	text	NO		number of distinct users that have posted the messages
lastUpdate	datetime	NO		last update of channel statistics

Table 12: chart_twitter_canali table

chart_twitter_retweet

Column	Type	Null	Default	Description
<i>id</i>	int(11)	NO		the row identifier (auto_increment)
request	text	NO		search text
count	int(11)	NO		number of messages relevant to search on a certain day

data	date	NO		date to which refers the count of messages
twitterUser	text	NO		number of distinct users that have posted the messages

Table 13: chart_twitter_retweet table

chart_twitter_retweet_canali

Column	Type	Null	Default	Description
id	int(11)	NO		the row identifier (auto_increment)
canale	text	NO		channel name
count	int(11)	NO		number of retweets related to the channel on a certain day
data	date	NO		date to which refers the count of messages
twitterUser	text	NO		number of distinct users that have posted the messages
lastUpdate	datetime	NO		number of messages relevant to channel on a certain day

Table 14: chart_twitter_retweet_canali table

chart_user

Tables "chart user", "chart user channels" are used for storing data necessary to display charts of users. Again this is done in the background and the data are retrieved for displaying of graphs and tables.

Column	Type	Null	Default	Description
id	int(11)	NO		the row identifier (auto_increment)
request	text	NO		search text
count	int(11)	NO		number of distinct users who have written at least a message that matches the search criteria

Table 15: chart_user table

chart_user_canali

Column	Type	Null	Default	Description
id	int(11)	NO		the row identifier

				(auto_increment)
canale	text	NO		channel name
count	int(11)	NO		number of distinct users who have written at least one message in the channel

Table 16: chart_user_canali table

crawling_stat

Tables "crawling_stat" and "crawling_stat_day" are used to store statistics on the process of crawling. Both tables have the same fields but the first contains the total values of the statistics and the second stores the statistics for a single day. Both are used to evaluate the efficiency of the system.

Column	Type	Null	Default	Description
id	int(11)	NO		
channel	varchar(128)	NO		channel to which are referred the statistics
tweet_db	int(11)	NO		Tweet number in the database
tweet_orig	int(11)	NO		Number of fathers tweets associated with retweets in the database
missingFather	int(11)	NO		Number of missing fathers tweet
fatherCoverage	float(11,2)	NO		Coverage of fathers Tweet
retweet_db	int(11)	NO		Retweet number in the database
retweet_orig	int(11)	NO		Retweet number declared by Twitter
retweetCoverage	float(11,2)	NO		Retweet Coverage
lastUpdate	datetime	NO		Date and time of the last statistics update
keyword_number	int(11)	NO	0	number of search associated with the channel
keyword_exec	int(11)	NO	0	Search number really executed
total_launched	int(11)	NO	0	Number of search launches performed
saturation_number	int(11)	NO	0	Number of saturations
saturation_perc	float	NO	0	Saturation percentage compared to the total

				number of launches
data	date	NO		Day the statistics relate

Table 17: crawling_stat table

crawling_stat_day

Column	Type	Null	Default	Description
<i>id</i>	int(11)	NO		
channel	varchar(128)	NO		channel to which are referred the statistics
tweet_db	int(11)	NO		Tweet number in the database
tweet_orig	int(11)	NO		Number of fathers tweets associated with retweets in the database
missingFather	int(11)	NO		Number of missing fathers tweet
fatherCoverage	float(11,2)	NO		Coverage of fathers Tweet
retweet_db	int(11)	NO		Retweet number in the database
retweet_orig	int(11)	NO		Retweet number declared by Twitter
retweetCoverage	float(11,2)	NO		Retweet Coverage
lastUpdate	datetime	NO		Date and time of the last statistics update
keyword_number	int(11)	NO	0	number of search associated with the channel
keyword_exec	int(11)	NO	0	Search number really executed
total_launched	int(11)	NO	0	Number of search launches performed
saturation_number	int(11)	NO	0	Number of saturations
saturation_perc	float	NO	0	Saturation percentage compared to the total number of launches
data	date	NO		Day the statistics relate

Table 18: crawling_stat_day table

hashtags

It contains hashtags related to individual tweets.

Column	Type	Null	Default	Description
tweetID	varchar(512)	NO		ID of the message on Twitter (refers to the field twitterId of the table ax_twitter)
hashtag	text	NO		name of hashtag in the message identified by tweetID

Table 19: hashtags table

mentions

It contains the users mentioned in the individual tweets.

Column	Type	Null	Default	Description
tweetID	varchar(512)	NO		ID of the message on Twitter (refers to the field twitterId of the table ax_twitter)
mention	text	NO		name of the user mentioned in the message identified by tweetID

Table 20: mentions table

process_list

It contains information on the processes that are running.

Column	Type	Null	Default	Description
ID	int(11)	NO		
name	varchar(128)	NO		process name
process_id	bigint(32)	NO		id of the process within the server
last_exec	datetime	NO		last execution of the process
status	tinyint(1)	NO	0	process status (active, disactive)
lastID	int(11)	NO		Last id of the table ax_twitter used (used only by scripts of charts)

Table 21: process_list table

tweet_retrieve

Table used to keep track of tweets to recover and to avoid requiring the ones that were not recovered after 3 attempts.

Column	Type	Null	Default	Description
<i>twitterId</i>	varchar(32)	NO		ID of the tweet
first_attempt	tinyint(2)	YES	0	outcome of the first attempt (1 not found, 2 recovered)
second_attempt	tinyint(2)	YES	0	outcome of the second attempt (1 not found, 2 recovered)
third_attempt	tinyint(2)	YES	0	outcome of the third attempt (1 not found, 2 recovered)

Table 22: tweet_retrieve table

users

The table "users" is used to manage users and to get a "role" to users. Associable roles are 3: Administrator, User, Viewer. Viewer can display only what the Users or the Administrator decides to share with him.

Column	Type	Null	Default	Description
<i>ID</i>	mediumint(9)	NO		user identification
username	varchar(60)	YES	NULL	Specifies the user name
password	varchar(60)	YES	NULL	Specifies the password encrypted with the MD5 algorithm
ruolo	enum('amministratore', 'utente', 'viewer')	NO		Indicates the role of the user (User, Administrator, Viewer)

Table 23: users table

variable

In table "variable" are stored values of the variables used by the process of crawling.

Column	Type	Null	Default	Description
cicle_time	varchar(128)	NO		Waiting time between two cycles
log_time	longtext	NO		How long to keep log files in memory (daily, weekly, monthly, always)
maxtweet_number				Maximum number of tweets per request to Twitter

Table 24: variable table

3.1.4 *Crawler*

The Crawler process is an independent thread that has an internal methods to construct the request and forward it to Twitter. As shown by the diagram in Figure 35 Crawler process functions are essentially 3:

- Taking charge of the first Search with deadline expired
- Formulation of the request to Twitter
- Posting new messages in the database

Can be instantiated a number of processes equal to the number of Twitter accounts required: account number is given by the Tweets peak is expected to reach.

Class Diagram

The class is called `TwitterThread` and derives from the PHP class `Thread`. The class `TwitterThread` has a dependency on the `DB` class that provides methods to connect to the database using the `PDO` class provided by PHP.

The `TwitterThread` class exposes the methods for extracting the hashtag, and the mention of the links from the messages and to differentiate from retweet and tweet. The input method of the thread is `run()`: inside there is the main loop that remains active until is detected the closing command of the thread (in this case is the presence of a file named "stop.txt"). The method responsible for making the request to Twitter and to extract information from the message to enter them in the database is `getTweets(account, text_search, id_req, lang)`. The parameters accepted by the function are the Twitter account to be used, the search text, the id of the research and the language to to search.

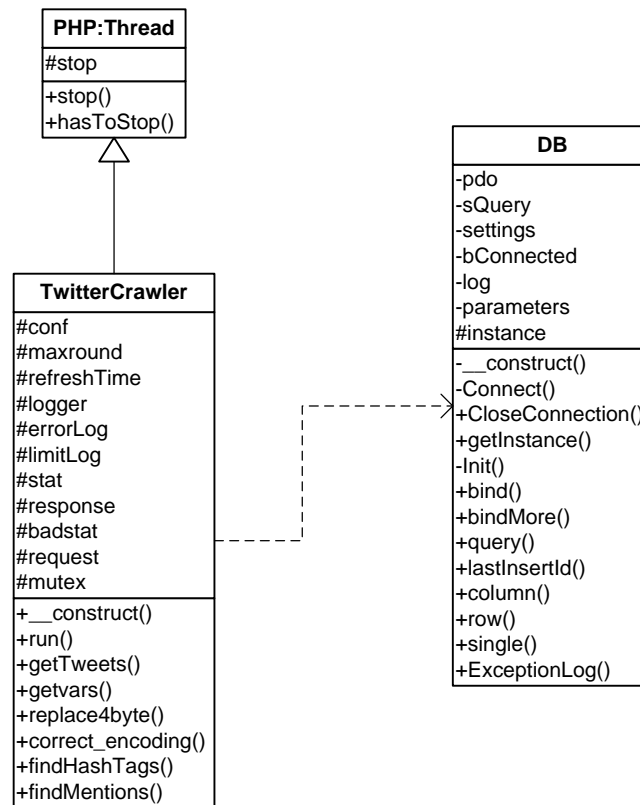


Figure 34: Class Diagram for Crawler process

State Diagram

Crawler processes are independent processes that are running in the background looking for a search that have the execution deadline expired and the status set to "waiting" in the table ax_request_twitter. As soon as a process it finds one, it blocks the table by placing a lock so that other processes can not write to and read from the table until the state of Search has been changed to "processing": this means that no other process can take in charge the same search. After this phase, the lock is removed from the table.

The next step of the process is to make the request to Twitter and send it by using the Search API "https://api.twitter.com/1.1/search/tweets.json". Each request can produce a maximum of 100 results, the process must include in database only those "new" and save the number in the table ax_request_twitter in the line corresponding to the search running: this time is not entered the lock because the search not can be handled by any other process until it changes its state.

On completion it changes the search status in "completed" and the Crawler process begins again its execution cycle.

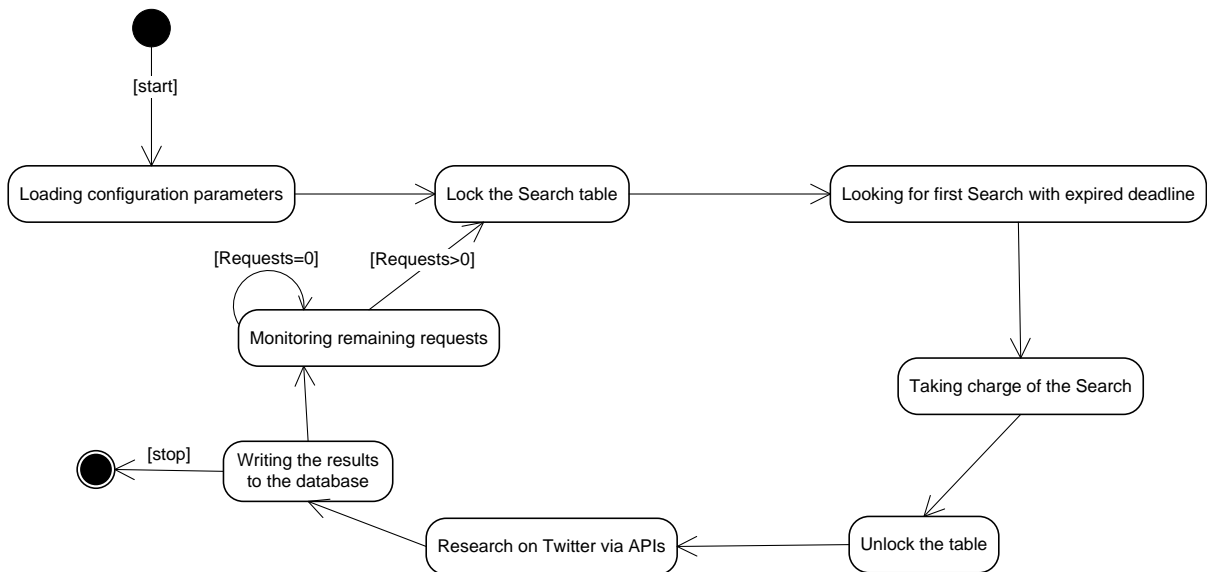


Figure 35: State diagram for TwitterCrawler

3.1.1.5 Scheduler

The Scheduler is the process that takes care of starting the Crawler processes and giving a priority to the Searches. The process is an independent thread that is always running and accesses the table "ax_request_twitter" in the database to read the state of Searches and changing priorities and deadlines based on the number of "new" tweets in the database in the last two executions of Search.

Class Diagram

The process scheduler is a thread independent. The class is called TwitterScheduler and derives from the PHP class Thread. The class TwitterScheduler has a dependency on the DB class that provides methods to connect to the database using the PDO class provided by PHP.

The method "run" is the entry point of the thread and is responsible for the activation/deactivation of Crawler processes and containing the cycle that remains active until it detects the command of "stop" and the thread is finished. Inside the loop are called the two main methods of the class: minQuery () and updateStatus ().

Function minQuery() is responsible for creating a minimal set of searches among active ones to reduce the number of requests to Twitter. The criterion of reduction is based on the interpretation of the operators used: the results of some search may be included in those of other more "generic" search:

1. all searches composed of a single word (hashtag, mention, free text and sender) are inserted directly into the subset without additional filtering

2. the multiple-word searches are sorted according to the criterion for which, if they contain a keyword of a search already present in the subset are discarded

By following these rules is possible to reduce the set of searches of 10%: even if it is not a particularly significant reduction but reduces the number of requests and allows other applications to have more margin for execution with respect to the limit imposed by Twitter.

Function `updateStatus()` is responsible for updating the priorities and deadlines associated with each Search.

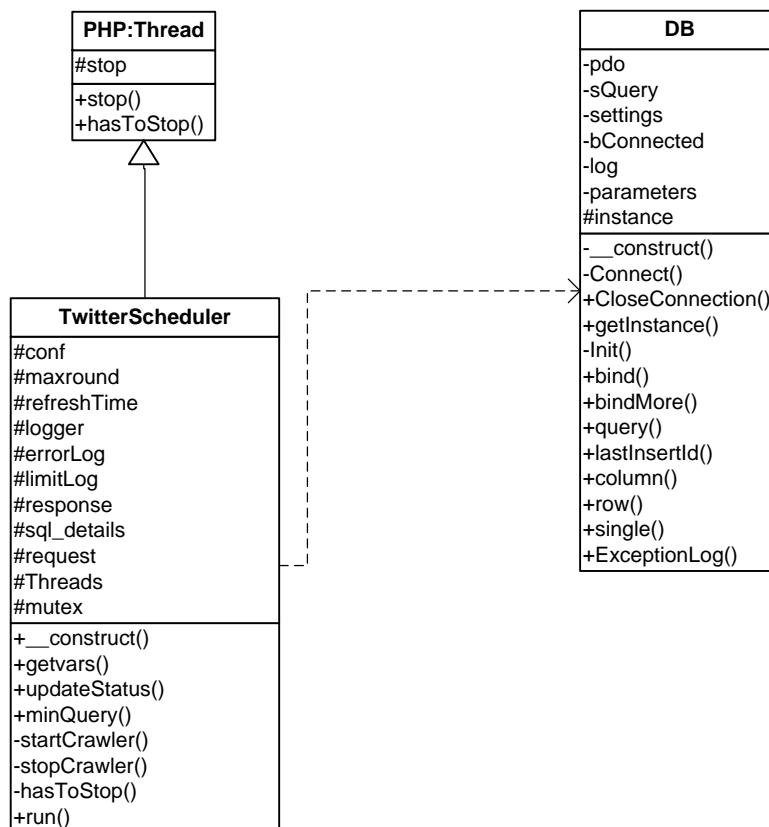


Figure 36: Class diagram for TwitterScheduler

State Diagram

In Figure 37 is shown the state diagram of the Scheduler process that lists all the actions that the process needs to complete every execution cycle. As can be seen from the diagram the first operation which is conducted by the process when is started, is initiating the Crawler processes: the number of processes to be activated is specified in a configuration file.

Once have started the Crawler processes, system enters in the main loop: initially takes place the reduction of the set of active searches. This operation is performed at every cycle because the set of active searches can change over time for both the addition of new search either for the deactivation / activation of the existing search.

The major action by the Scheduler is updating the priorities and deadlines of execution of the searches. The deadline shall be updated in relation to the priorities calculated based on the average of the results of the last two runs of the search: the results of the search are the number of "new" messages in the database. At each cycle are updated only the priorities and deadlines of the searches whose status is set to "completed": after upgrading, the status of the search is brought to "waiting" and may be taken over by a Crawler process when the deadline expires.

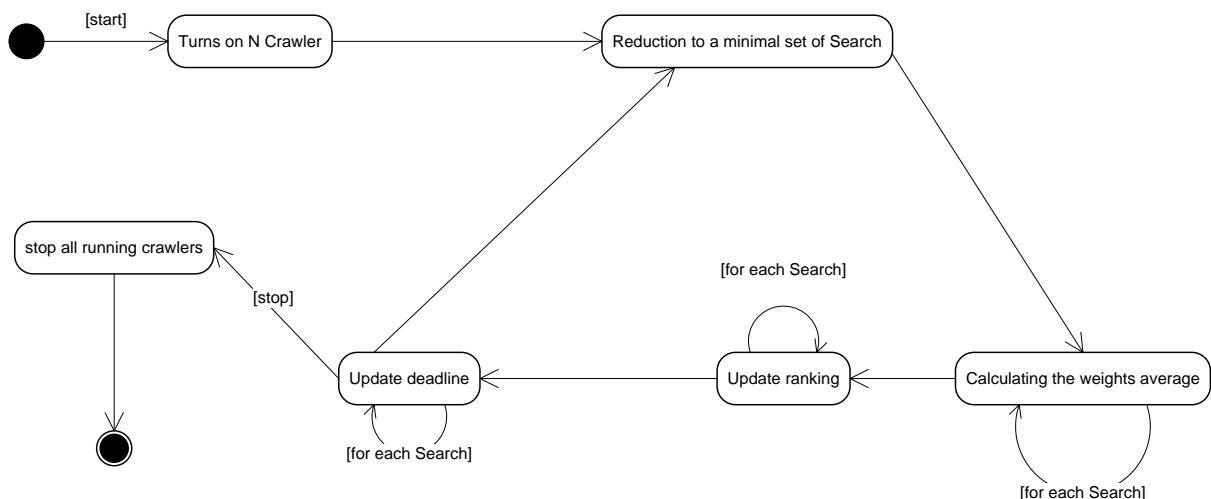


Figure 37: State diagram for TwitterScheduler

State Diagram for ranking update

In Figure 38 is shown the statechart for updating of the priorities and deadlines of the searches. The update is carried out for all searches that are set to "completed". As previously mentioned these depend on the average of the results of the last two executions of the search. From this average depends updating policy of priorities that have values between 5 and -160:

- if media=100 and is the first run of the Scheduler then the priority is set to 0;
- if media≤20 there are 2 cases:
 - if the priority is equal to 5 is left unchanged
 - if the priority is different from 5 is increased by 1
- if 20<media≤35 there are 2 cases:
 - if the priority >= 0, this is set to 0
 - if the priority <0, this is increased by 1
- if 35<media≤60 the priority is left unchanged;
- if 60<media≤75 there are 2 cases:
 - if the priority is different from -159 is decreased by 1
 - if the priority is equal to -159 is left unchanged

- if media>75 there are 3 cases:
 - if the priority is greater than -4, this is set to -4
 - if the priority is between -4 and -159:
 - if media=100 and rank*2>=-159 the priority is set to twice the current
 - otherwise the priority is decreased by 1
 - if the priority is equal to -159, this is left unchanged

After updating the priority, the execution deadline must be updated on the basis of the new priority:

1. corresponds to 16 minutes of waiting which is the reset time of the limit of searches of Twitter
2. values greater than 1 double the waiting time
3. 0 corresponds to 8 minutes
4. negative values reduce the waiting time according to the following condition $8 / (|k| + 1)$ where k is the ranking in absolute value

So according to this algorithm, the waiting time before the expiration of the new deadline varies between a maximum of 256 minutes (just over 4 hours) to a minimum of 3 seconds. In this way the system is able to adapt to variations in the frequency of the messages being able to follow either slow that explosive events. Moreover for slow events is able to reduce the number of requests in a consistent way.

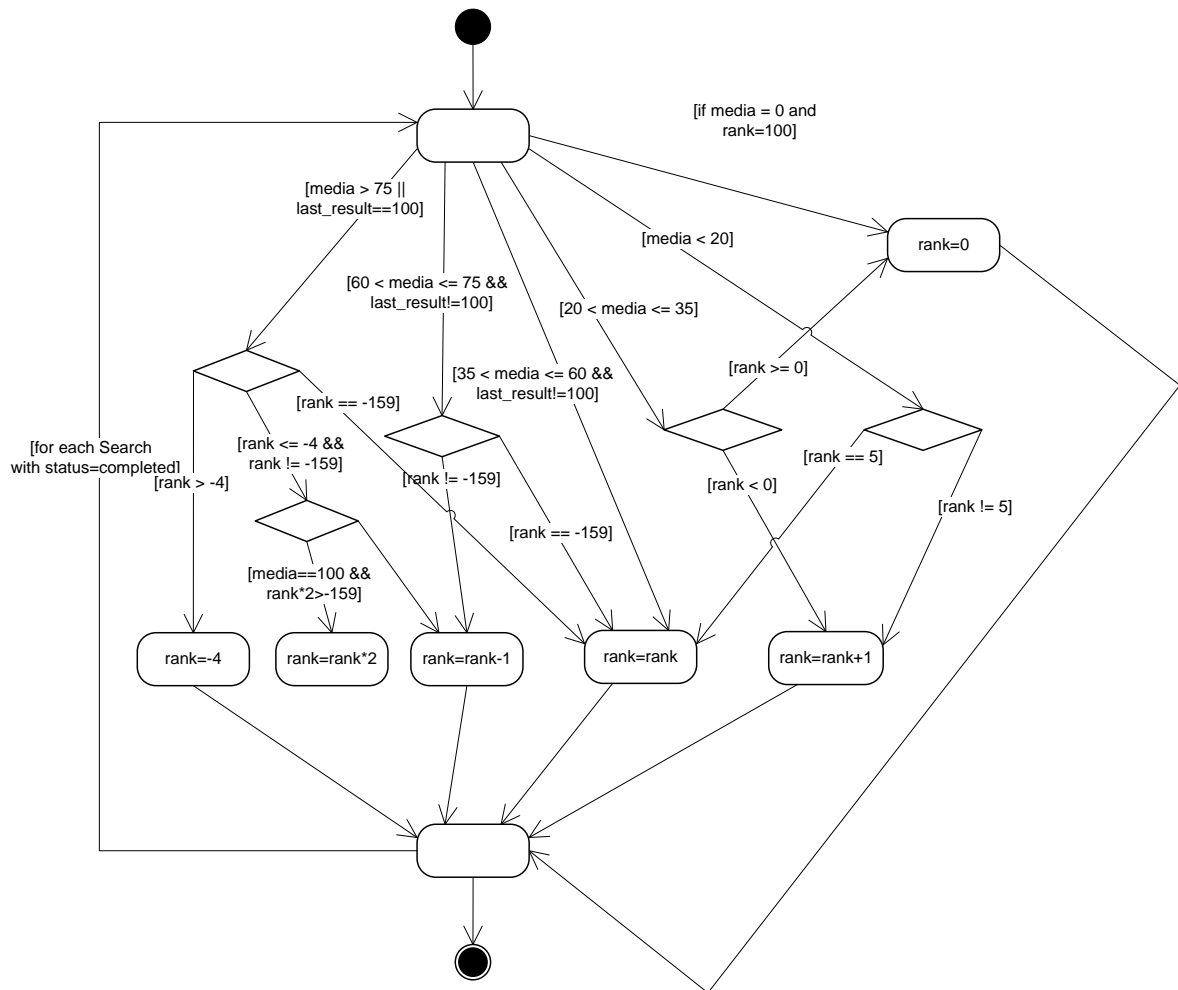


Figure 38: State diagram for ranking update (*rank* is the priority associated with Search, *media* is the average of the results of the last 2 execution of Search)

3.1.6 Other Scripts

PHP script to update the tweet in the database

The process for updating the data of the tweet is responsible to retrieve the updated information about the messages in the database by making new requests to Twitter on the basis of the ID of the tweet. The API used is: <https://api.twitter.com/1.1/statuses/lookup.json>. At this API can be passed up to 100 IDs of tweets a time. The update is done mainly to update the number of retweets and favorites. The process must use an account for the Twitter API recovering from the table "account_twitter" of the DB from the available (status "available"). The only action of writing that is done in the "ax_twitter" Database is the UPDATE of tweets for which Twitter has provided an answer. The process writes in the "processlist" table its status: at the start of process is written the start date and time and the status is changed to "running"; at the end of the process, the status is updated in "idle". The process is scheduled once a day.

PHP script for calculating the number of tweets\retweets per day per channel\search

This script takes care of calculating statistics on the number of tweets and retweets per channel \ search divided by day: this data is stored in MySQL database tables (chart_twitter_retweet, chart_twitter_retweet_canali, chart_twitter and chart_twitter_canali). This serves to make the display of graphics faster. The process writes in the "processlist" table its status: at the start of process is written the start date and time and the status is changed to "running"; at the end of the process, the status is updated in "idle". The process is scheduled once a day.

PHP script for calculating Crawling statistics

This script takes care of calculating statistics for the phase of Crawling: in particular is calculated the percentage of coverage of father tweets of retweets in the DB and the number of saturations of search requests. For statistics on the father tweets is scanned the database to identify the original tweet of retweets that are not present in database. To find the number of saturations are analyzed log files of the process of crawling. Saturation is identified when a request to Twitter provides 100 new posts to be included in the database. The process writes in the "processlist" table its status: at the start of process is written the start date and time and the status is changed to "running"; at the end of the process, the status is updated in "idle". The process is scheduled once a day.

PHP script for the recovery of missing father tweets

This script takes care of the request to Twitter the missing fathers tweet: these are the original tweet of the retweets. To identify tweets to require first are retrieved original tweets of the retweets in the database and then checks whether these tweets are present or not in the database. The missing tweets are requested to Twitter: the request is made by using the ID of the tweet. The API used is: <https://api.twitter.com/1.1/statuses/lookup.json>. At this API can be passed up to 100 IDs of tweets a time. The process writes in the "processlist" table its status: at the start of process is written the start date and time and the status is changed to "running"; at the end of the process, the status is updated in "idle". The process is scheduled once a day.

3.2 Dashboard

3.2.1 Requirements

1. The system must have a table for each channel in which are inserted active users on that channel.
2. The system should allow the user to enter the research to be carried out through a form.

3. The system should display only the channels associated with the user.
4. The form for entering the search must contain:
 - a. a field for entering the search text (hashtags, keywords or persons)
 - b. a field for entering the search language
5. Searches can be used for one or more channels.
6. User can create a channel and associate the existing research to that channel. If the search does not exist the user must enter a new search.
7. Searches should be presented in tabular form with the following fields:
 - a. ID
 - b. Status
 - c. Text
 - d. Language
8. The name of the channel must not be an empty string or to have the same name already associated with the user channels.
9. The system must include a form for inserting data for Twitter access. This form must contain:
 - a. a field for entering dell'OAuth Access Token
 - b. a field for entering the Secret Access Token
 - c. a field for the insertion of the Consumer Key
 - d. a field for entering the Consumer Key Secret
10. The system should display statistics on retrieved messages through graphics.
11. The system should create two levels of statistics: statistics for administration and statistics for user.

User Requirements

1. User should only see the channels that are associated with its ID.
2. User must be able to enter new channels.
3. User must be able to edit and delete their own channel.
4. User must be able to enter new Searches.
5. User should not be able to edit or delete searches.
6. User can assign to channel any search already entered into the system by other users.
7. User must display only messages relating to the Searches associated with its channels.
8. User must see only charts on its channels.
9. The system shall allow the user to display the trend in terms of number of tweets of all the channels associated with the user in a specified time interval [time window displayed is selectable on the graph]
10. The system must report the user in tabular form all the channels he owns and the searches that compose them.

11. For each channel, the system should allow the user to view a page that:
 - a. presents in tabular form a summary of the Twitter users that appear in highlighting the research activities within the selected channel.
 - b. presents the trend in terms of number of tweets of the selected channel in a specified time interval [time window displayed is selectable on the graph]
 - c. must provide for a system to highlight some events that took place in the Timeline to make visible the correlation with the number of tweets collected.
 - d. must allow the user to display the trend in terms of number of tweets of all the searches associated with the channel in a specified time interval [time window displayed is selectable on the graph]

Administrator requirements

1. The administrator will have to see all the channels included in the system.
2. The administrator must be able to enter new channels.
3. The administrator must be able to modify and delete any channel.
4. The administrator must be able to enter new Search.
5. The administrator must view the list of all the Searches entered into the system.
6. The administrator must be able to edit or delete searches.
7. The administrator can assign to any channel research already in the system.
8. The administrator can view the graphs for all channels.
9. The system must make it possible to select to display the trend of all Channels in terms of number of tweets present in the system in an interval of time [the time interval displayed on the graph is selectable]
10. The system must represent the number of tweets collected over the entire time period for channel via histogram. For each channel of the histogram through a transaction drilldown you can access a sub-histogram that presents the number of tweets for every Search in Channel.
11. The system shall provide in tabular form a summary of the Twitter users that appear in the Search highlighting their activities within the channels.

3.2.2 Frontend Architecture

The front end is designed to be used either by the User that the Administrator: to accomplish that are designed two different architectures for the two types of users. Moreover has been included a display of some parts of the system also to unauthenticated users, which is simply a derivation of the display designed for authenticated users. As for the user side, the front end must allow to manage the channels associated to the User and see some statistics about the messages downloaded primarily in graphical form. As for the part of Administrator, the frontend must allow the management and viewing statistics for all channels and all searches included in the system, checking the status of background

processes and viewing statistics on background processes (in particularly for the process that queries on Twitter).

User Frontend

When you enter the front end by unauthenticated or authenticated user the first page that appears is the page "Channel Statistics": This page shows the list of "public" channels both in the form of a graph and in table form (Figure 40). In the column Detail there are two buttons: the first on the left makes access to the detail page of the selected channel, the second is disabled and in the future will show the NLP analysis of tweet of the channel.

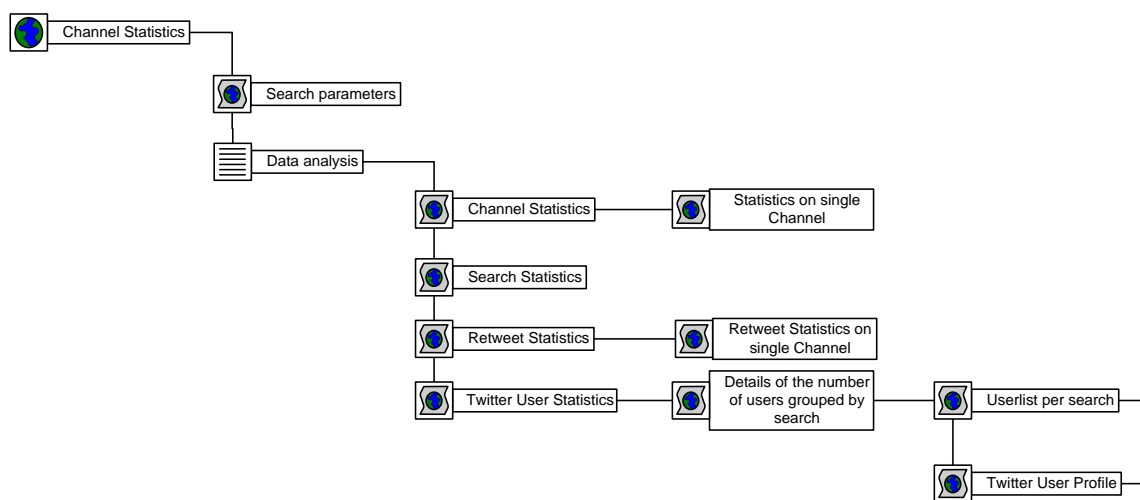


Figure 39: Structure of User Frontend

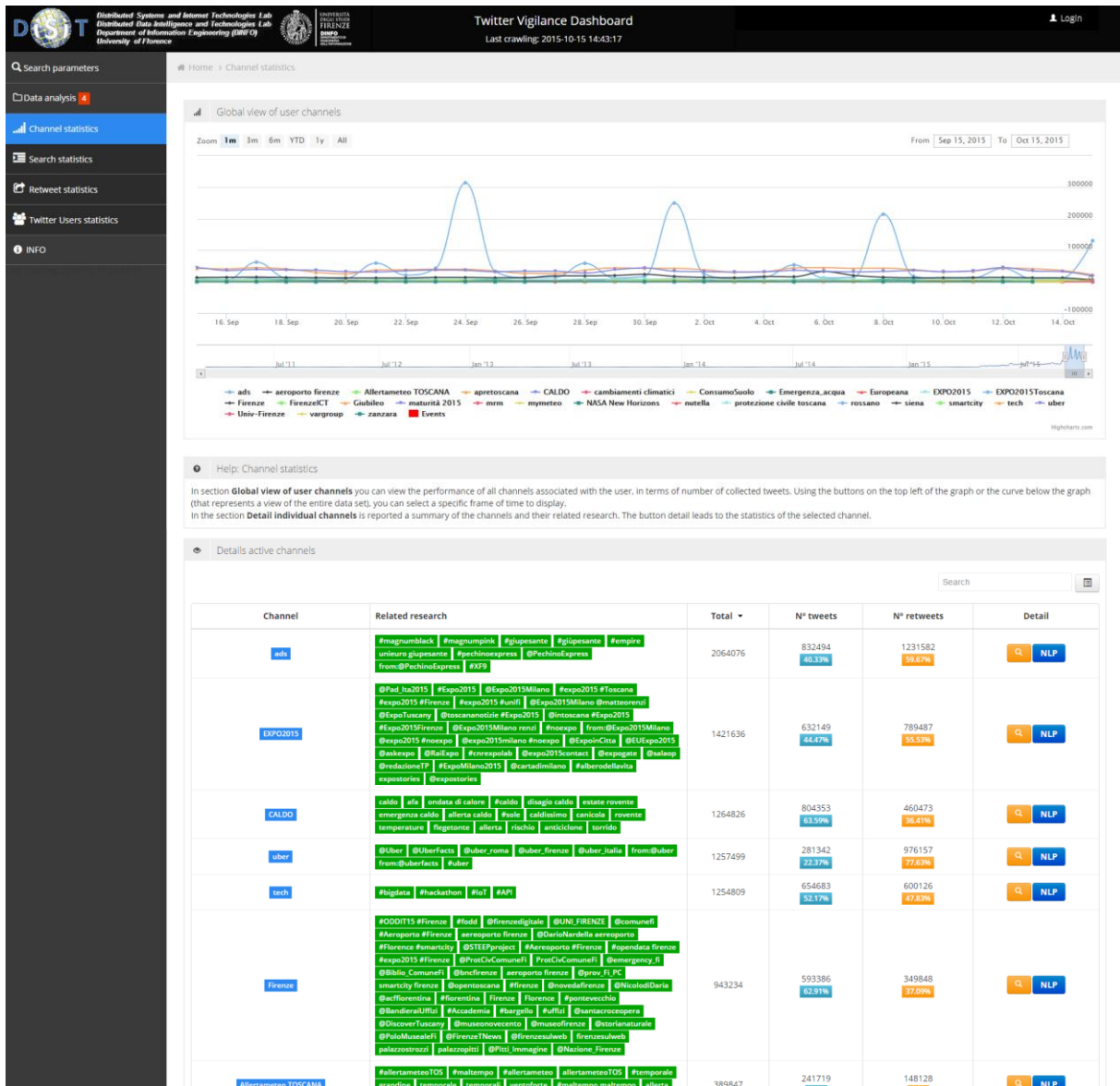


Figure 40: Channel statistics page

The graph shown in Figure 40 shows the number of tweets per day for each channel associated with the user. For displaying this graph was prepared the table "chart_twitter_canali" in the database in which are stored the necessary data by a separate process which will be described later. This table has updated at regular intervals by adding only the values for the downloaded messages since the last update.

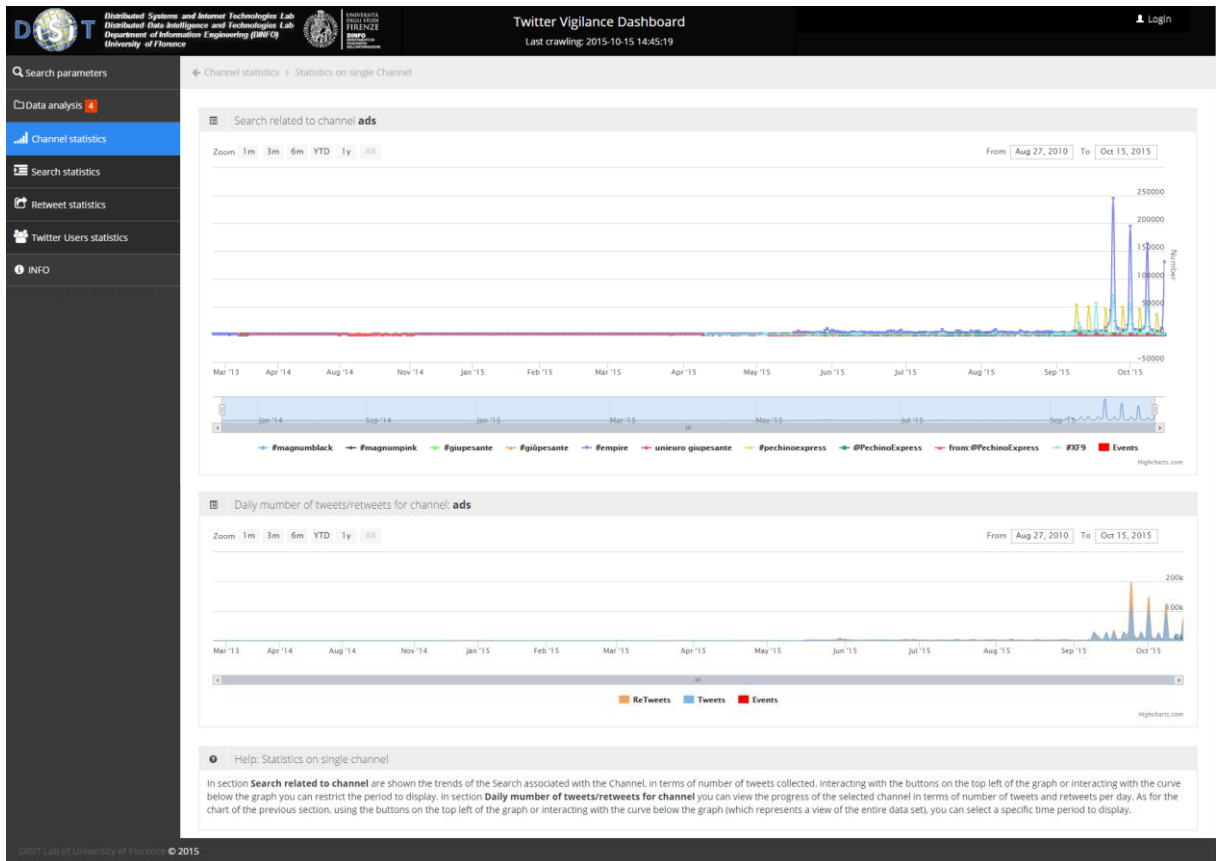


Figure 41: Statistics on single Channel page

Details page of the channel, Figure 41, shows two graphs: the top one shows the number of tweets per day for each search associated with the channel, the second shows the number of tweets and retweets per day for the entire channel.

For the display of this graph was prepared the table "chart_twitter" in which are stored the necessary data by a separate process which will be described later. This table has updated at regular intervals by adding only the values for the downloaded messages since the last update.

Search Statistics

On this page (Figure 42) you can see a histogram that shows the total number of messages for each channel.

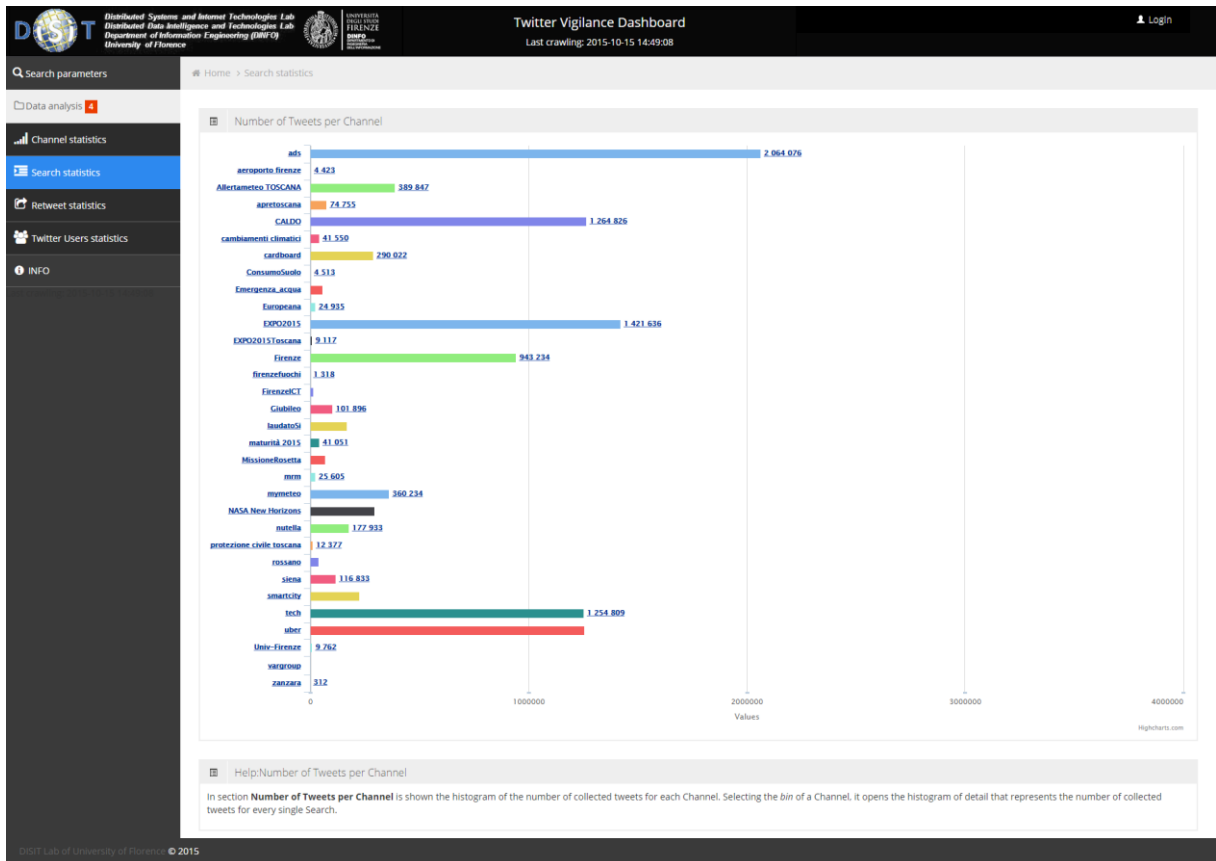


Figure 42: Search statistics page

To display this graph has used table "chart_twitter_canali" performing a sum of the values for each channel. Clicking on a bin it is possible display another histogram that shows the total values of the messages of the individual searches that are part of channel, as shown in Figure 43.

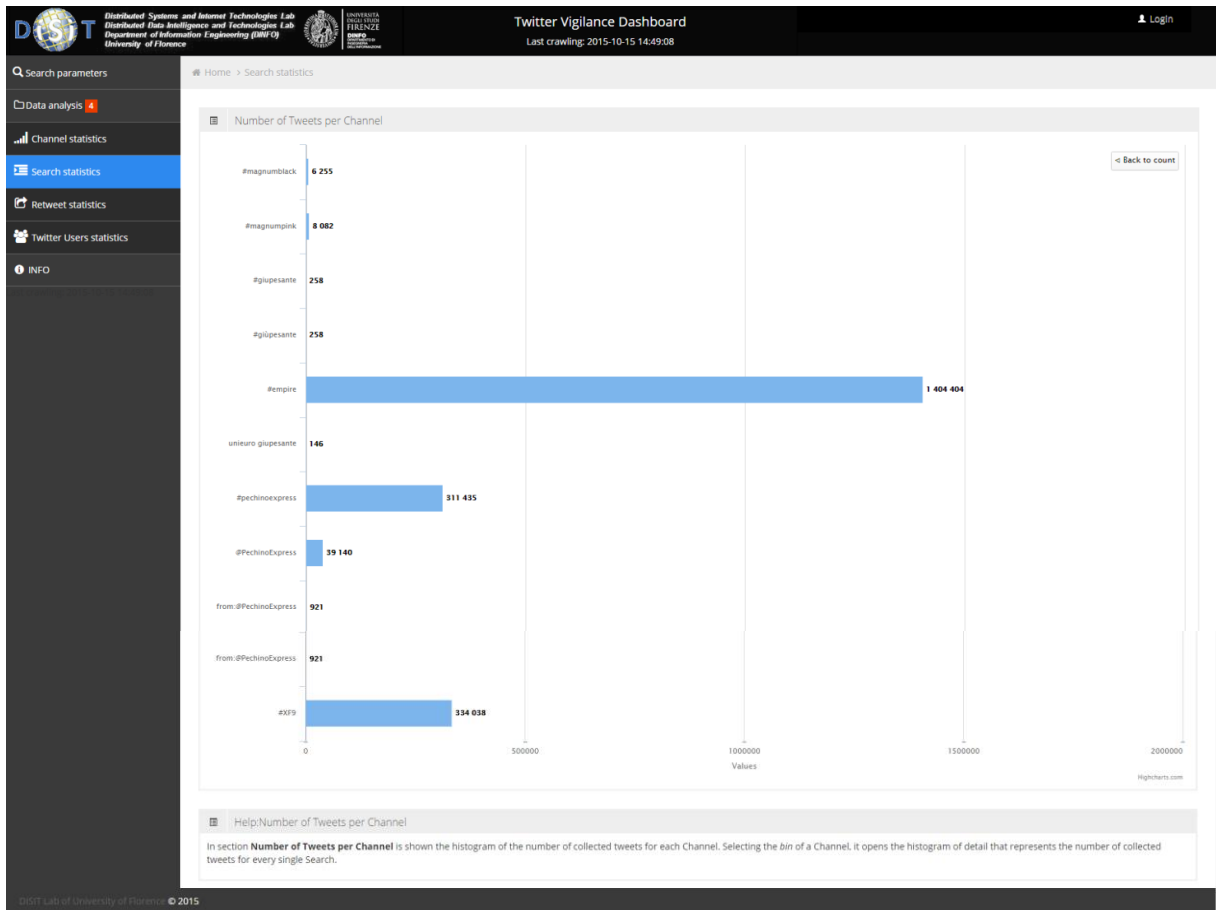


Figure 43: Search statistics page: single channel details

To display this graph has used table "chart_twitter" performing a sum of the values for each Search.

Twitter User Statistics

In this page (Figure 44) there are a histogram and a table representing the same data, ie the total number of distinct users for each channel.

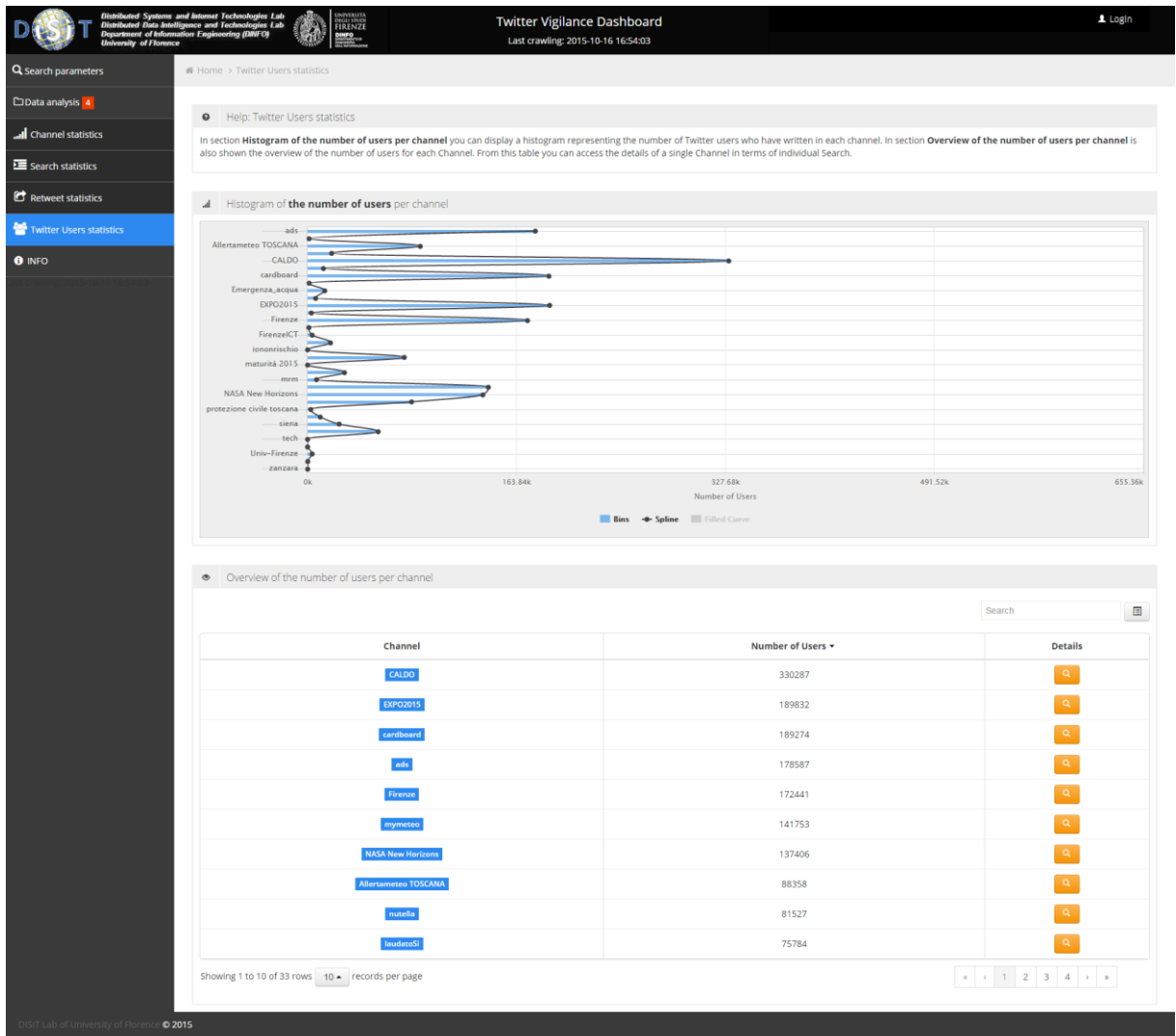


Figure 44: Twitter Users statistics

To display this graph and the table is used the table "chart_user_canali" that stores the total number of distinct users who have written at least one message that is part of channel.

Clicking on the button in the Details column of the table leads to the detail page of the channel, Figure 45, where there are two graphs and a table: The table represents the number of messages related to individual searches for that channel, the histogram shows the top 10 users with the greatest number of posts on the channel and the pie chart shows the distribution of users in individual research belonging to the channel.

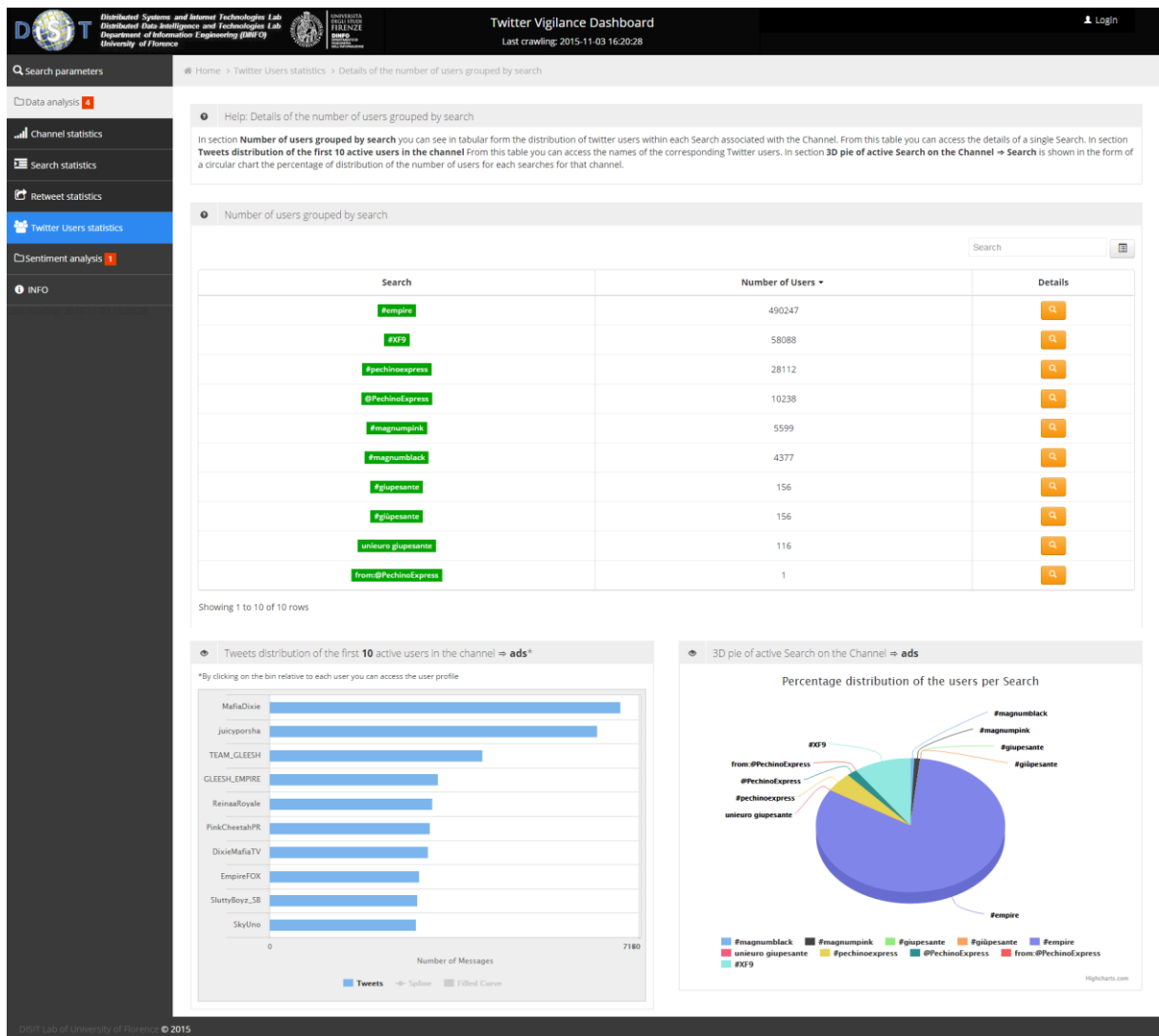


Figure 45: Details of the number of users grouped by search

For displaying histogram of users is used the table "chart_twitter_user_details_canali" that stores the authors of at least one message belonging to the channel and the number of posts.

To display the pie chart is used table "chart_user" that stores the total number of distinct users for each search.

Selecting the bin of a user it's possible to access the profile of the user (Figure 47).

Clicking on the button in the Details column of the table leads to the detail page of the Search, Figure 46, where is displayed a table: the table represents the list of users who have written at least one message among those recovered from the Search and the number of posts written by each user.

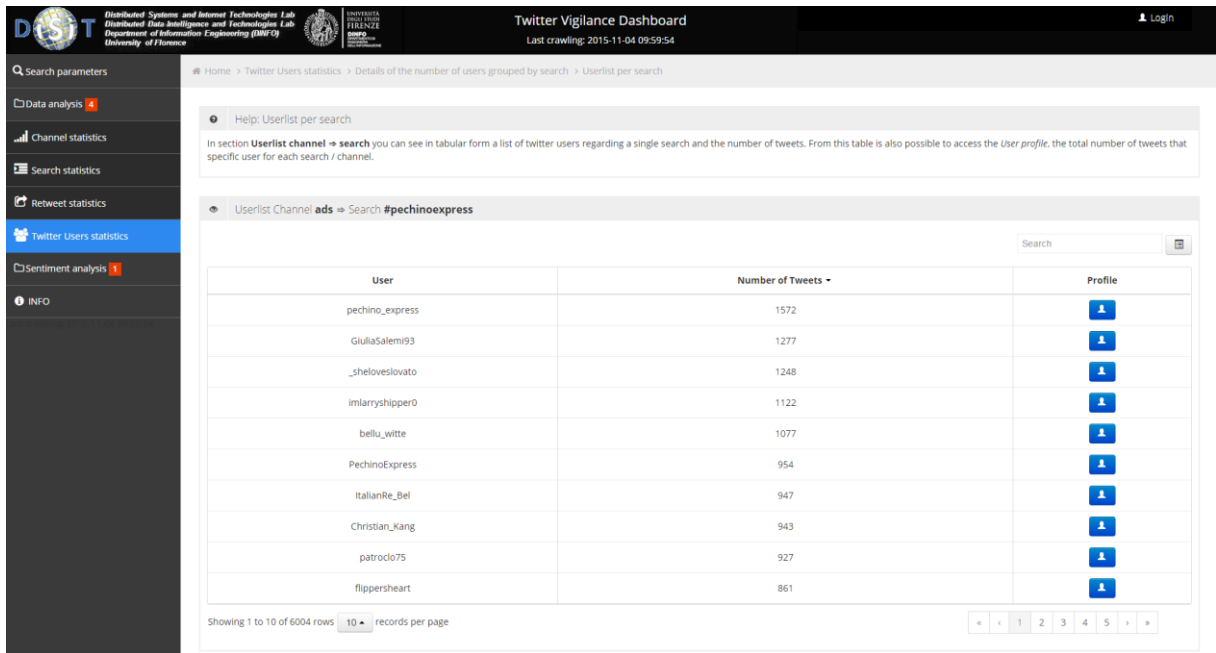


Figure 46: Details of user grouped by Search

Clicking on Profile button it's possible to access the profile of the user (Figure 47).

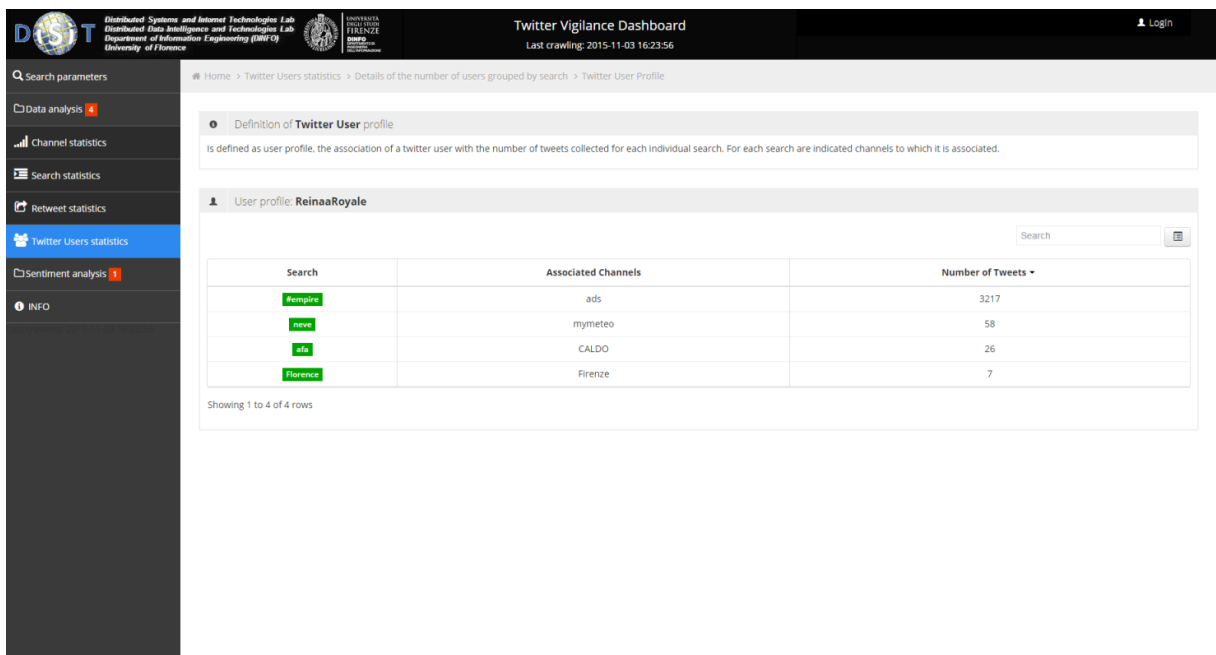


Figure 47: User profile

Page represented in Figure 47 displays the association of a twitter user with the number of tweets collected for each individual search. For each search outlines the channels to which it is associated.

To view this page is used the table "chart_user_details" in which users are stored authors divided by research and the number of posts.

Retweets

This page displays a bar chart that shows the number of tweets and retweets for each channel. To show this chart are used tables chart_twitter_canali and chart_twitter_retweet_canali.

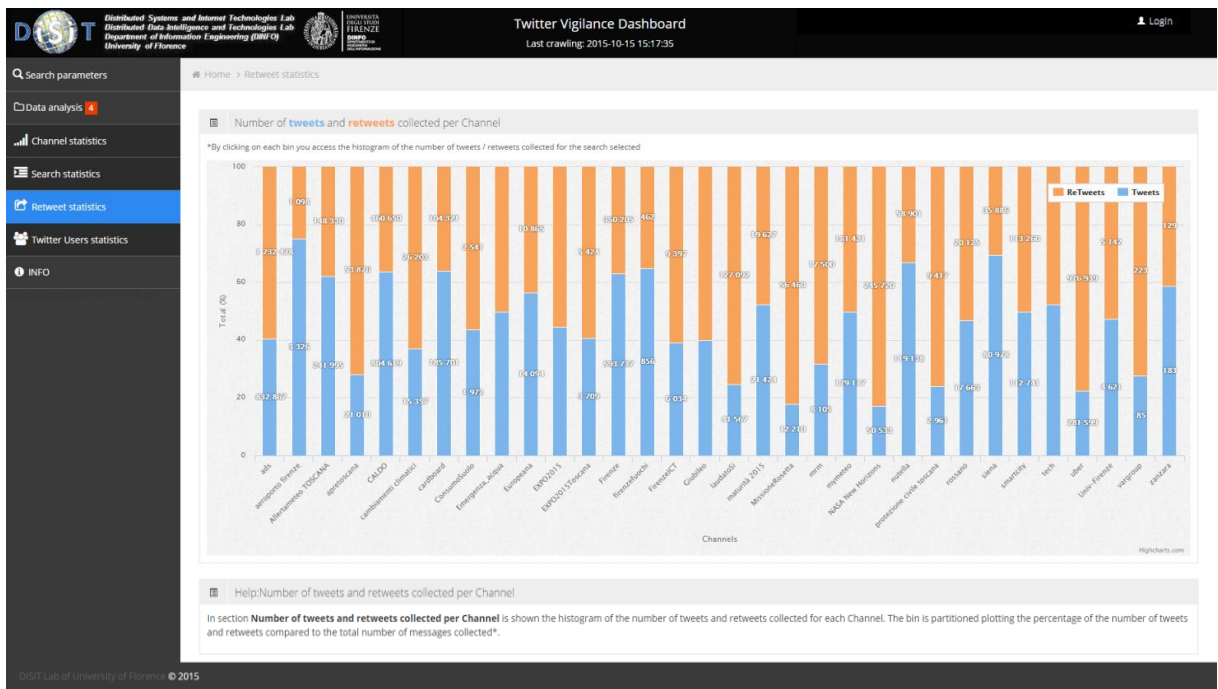


Figure 48: Retweet statistics page

Clicking on a single bar is shown the details of the research associated with the selected channel. To view the graph are used tables chart_twitter and chart_twitter_retweet.



Figure 49: Retweet statistics for single Channel page

Administrator Frontend

Respect to the User part, some parts remain unchanged while there are others for the management of the backend of the platform. The pages that are part of the Data Analysis section are identical to those of the User but instead of representing only channels associated to the user, in the case of Administrator are present all channels inserted in the platform.

Even the page "Search parameters" has the same features. As seen in Figure 51 there are two tables: the first shows the channel list, the second the list of searches. In the case of the channel table, the user will only display his channels while the Administrator all those present in the platform. The Search table can be viewed by the Administrator with the ability to edit and delete. The channel table contains buttons to edit, delete and put in standby channels.

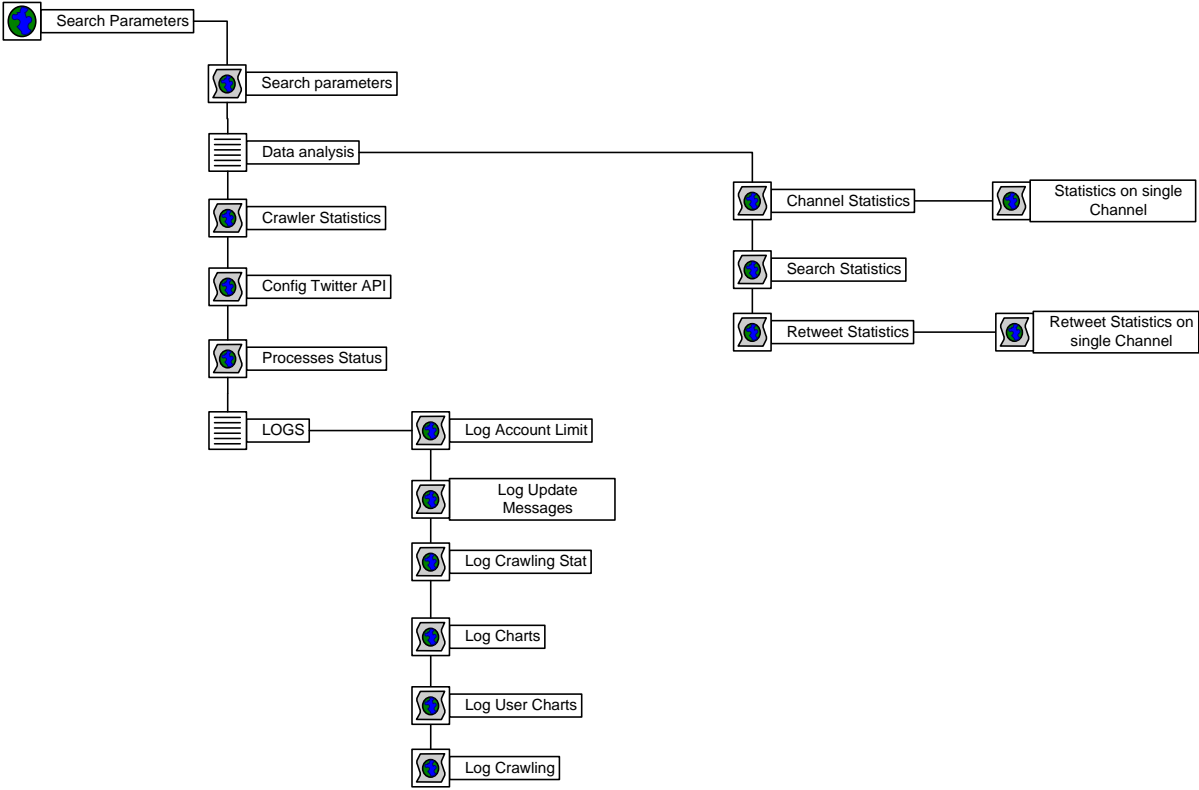


Figure 50: Structure of Administrator Frontend

Twitter Vigilance Dashboard
Last crawling: 2015-10-15 15:23:29

Channels

Id	Name	User	Last Statistics Update	Actions	Shared With
1	retto	nesi	2015-10-15 15:00:01	Turn ON	
2	aeroporto firenze	nesi	2015-10-15 15:00:01	Turn OFF	guest
3	EXPO2015Toscana	nesi	2015-10-15 15:00:01	Turn OFF	guest
4	eclap	nesi	2015-10-15 15:00:01	Turn OFF	
5	apretoscana	nesi	2015-10-15 15:00:01	Turn OFF	guest
6	zanzara	crisci	2015-10-15 15:00:01	Turn OFF	guest
7	protezione civile toscana	nesi	2015-10-15 15:00:01	Turn OFF	guest
8	Univ-Firenze	nesi	2015-10-15 15:00:01	Turn OFF	guest
9	chi parla di meteo	grasso	2015-10-15 15:00:01	Turn OFF	
10	ConsumoSuolo	grasso	2015-10-15 15:00:01	Turn OFF	guest

Showing 1 to 10 of 48 rows | 10 records per page

Searches

Id	Status	Channels	Text	Language	Last Execution	Actions
202	Active	ads	#empire		2015-10-15 15:23:29	
480	Active	Emergenza acqua	#drought		2015-10-15 15:23:01	
222	Active	smartcity	smartcities		2015-10-15 15:22:51	
394	Active	Smart Drugs	cannabis		2015-10-15 15:22:40	
414	Active	Allermeteo TOSCANA Maltempo MeteoAlert mymeteo	nubifragio	it	2015-10-15 15:22:39	
158	Active	CALDO	caldo	it	2015-10-15 15:22:39	
433	Active	mrm	@meeting_eng		2015-10-15 15:22:34	
294	Active	Maltempo	maltempo	it	2015-10-15 15:22:32	
252	Active	apretoscana	@EU_eGov		2015-10-15 15:22:31	
280	Active	Maltempo	pioggia	it	2015-10-15 15:22:07	

Showing 1 to 10 of 480 rows | 10 records per page

DSIT Lab. of University of Florence © 2015

Figure 51: Search parameters page

Via the link "Add Channel" opens a section that allows you to enter a new channel (Figure 52): required fields are the name of the channel and the list of the searches to associate with. There is the possibility to publish the channel sharing it with the "guest" user. Also via the link "Add Search" leads to a form for entering a new search that is automatically assigned to the new channel. The input section of a new channel has the same characteristics as that for editing of the channel.

Twitter Vigilance Dashboard
Last crawling: 2015-10-15 15:23:29

Home > Search parameters

Parametri ricerca

Crawler Statistics

Config Twitter API

Analisi dati 3

LOGS 8

Processes Status

INFO

New Channel

Channel name:

Share with:

Available Searches

ID	Status	Text	Language (ISO 639-1)
1	Active	#meteo	it
2	Disactive	#previsionimeteo #Firenze	it
3	Disactive	#meteo #veve #Firenze	it
6	Active	#ODDIT15 #Firenze	it
7	Active	#fodd	it
8	Active	#OpenDataDay #Firenze	it
9	Active	@flash_meteo	it
10	Active	@firenzedigitale	it
11	Active	@UNI_FIRENZE	it
12	Active	@apretoscana	it

Showing 1 to 10 of 480 rows | 10 records per page

+Add Search

Save Channel Cancel

Figure 52: Search parameters page -> add new channel

New Search

Text:

Channel:

Language: ALL

Save Search Close

Figure 53: Add new Search form

In Figure 54 is shown the page with the statistics of crawling: there are two tables, one for the total summary of the statistics and the daily details. The statistics shown are divided per channel and include:

- Number of Tweets
- Number of fathers Tweets
- Number of missing fathers Tweets
- Coverage of fathers Tweet
- Number of Retweets
- Number of Retweets declared by Twitter
- Coverage of Retweets
- Number of searches for that channel

- Number of searches performed
- Number of requests made to Twitter
- Number of saturations
- Percentage of Saturations

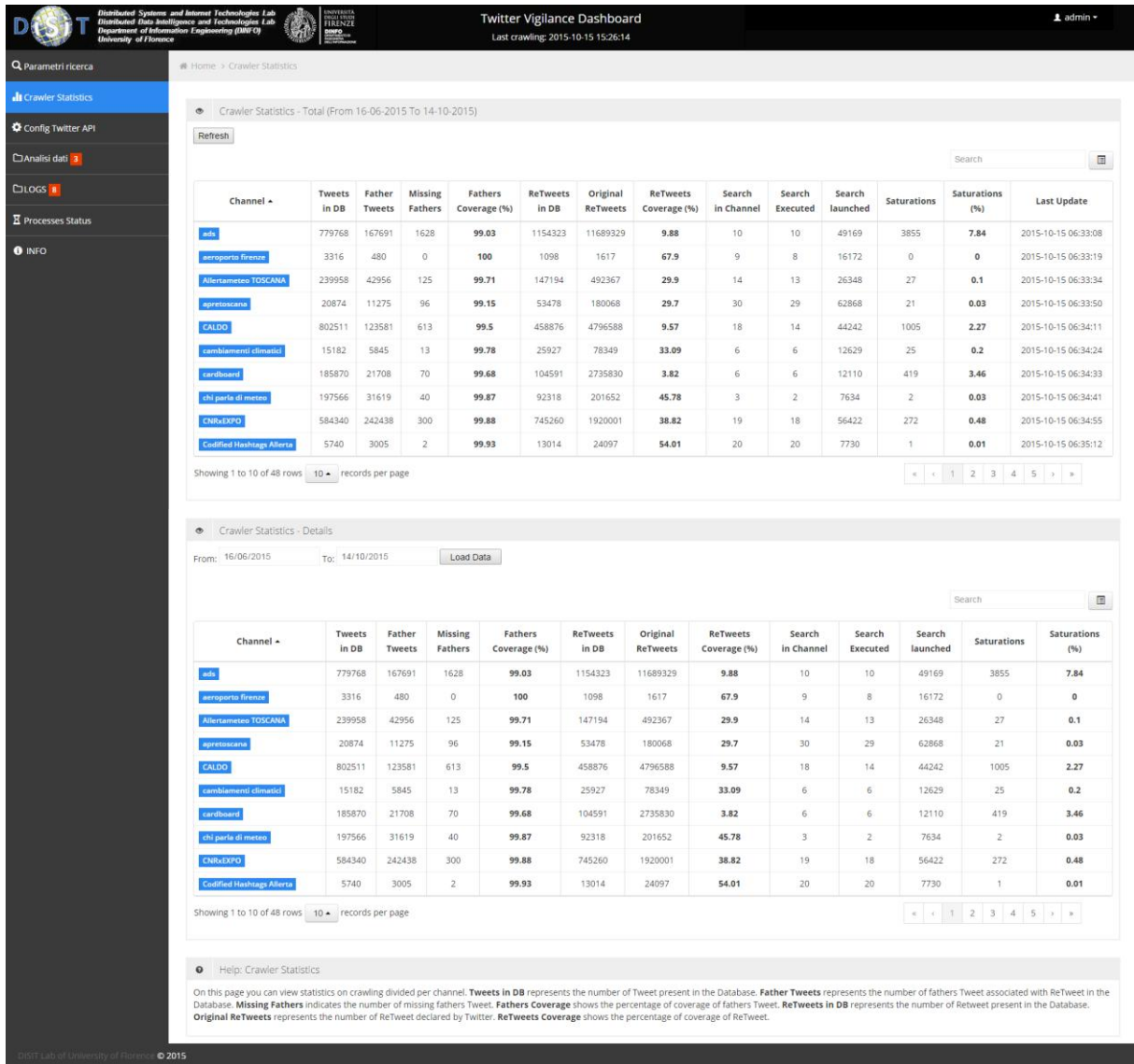


Figure 54: Crawler statistics page

In the page shown in Figure 55 is displayed all active processes on the backend and their status: in particular displays the job status (Running, Idle), the process ID and the date and time of the last execution of the process.

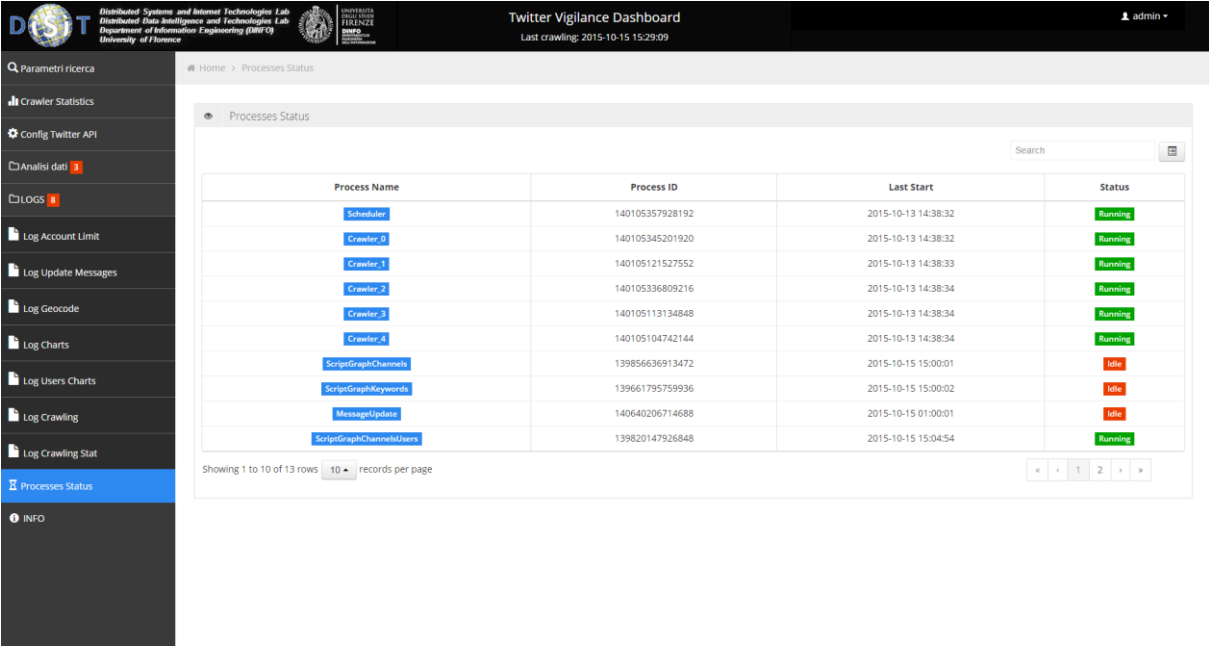


Figure 55: Processes status page

4 Testing and Validation

The main issues that were addressed in this project have already been highlighted in section 2.3. The validation will be performed only on the process of crawling which is the basis of the entire system because the display part is used merely to provide evidence of the results obtained. For this type of system validation occurs through the analysis of the execution of the system. In particular it was analyzed the functioning of the whole platform over the past 6 months. In this time period were collected more than 38 million messages (18 million retweets and 20 million tweets).

The system architecture that is running provides 5 Crawler processes each associated with a Twitter account. Each of these processes is monitored taking a log of the number of requests consumed / remaining.

The system includes 52 channels of which 47 are active and 517 searches of which 451 active. Searches that are not active are due to the fact that they are not associated with any channel or only to disabled channels. In fact, only the research that belong to at least one active channel are considered active.

In Table 25 are shown some values summarizing system performances. As you can see the main parameters are the percentage of coverage of the fathers tweets and the percentage of saturation. The fathers tweets are the original tweets associated with retweets stored in the database. Saturations occur when the system is unable to accommodate the growth in the number of messages that are available to a given search: in fact occur when a request produces 100 new posts of 100 returned.

The coverage of the fathers tweets provides a qualitative parameter on the system's ability to recover all the tweets, while the number of saturations indicates whether the system is able to adapt to changes in the amount of recoverable tweet.

Original Tweets in DB	Father Tweets	Missing Fathers	Fathers Coverage (%)	Active Search	Search Executed	Search launched	Saturations	Saturations (%)
19913684	3800595	53250	98,60	517	433	1580452	41075	2,60

Table 25: Crawling statistics

As shown in Table 25, the coverage ratio of the fathers tweets is excellent and fully respects what is specified in the requirements. In fact is recovered 99% of the fathers tweets: those

missing can be counted against the limits of the index of messages provided by Twitter. It will never be possible to retrieve all messages that are part of Twitter because only a portion is made available through the API.

The performance in last 6 months are summarized by the number of Tweets downloaded per day: as shown in Figure 56 there was a peak of more than 830,000 tweets per day and an average of 300,000 tweets per day in the last 3 months.

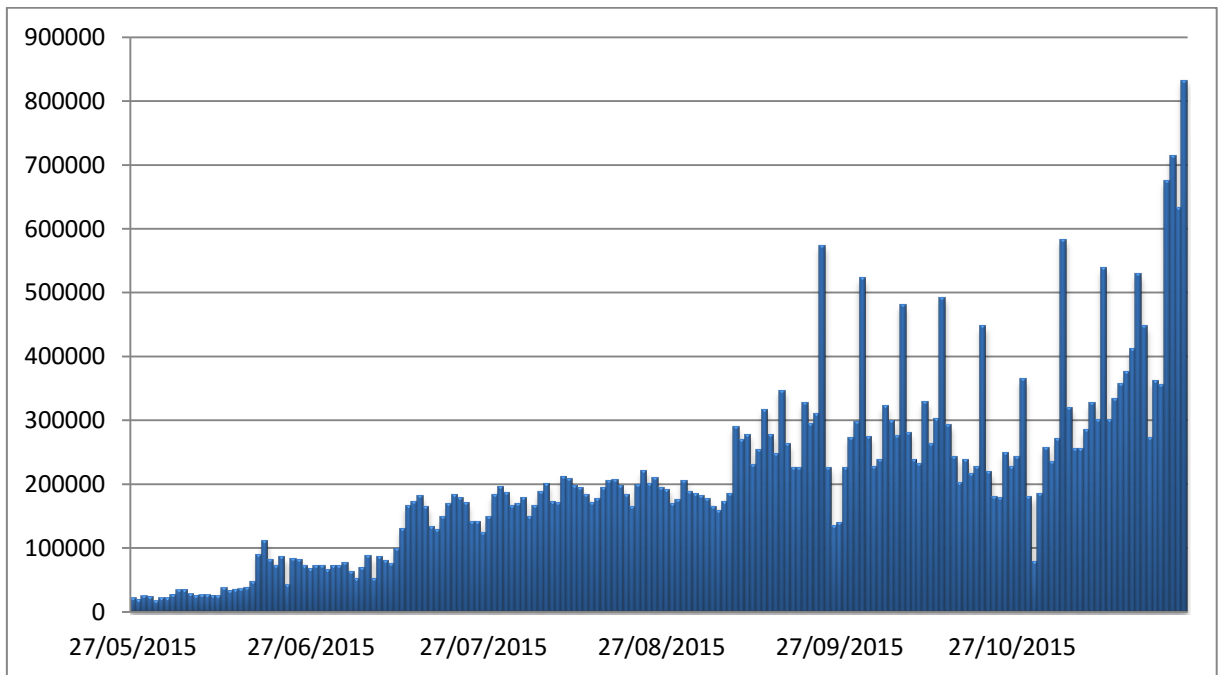


Figure 56: Number of Tweets downloaded per day

This threshold can be widely exceeded for the simple reason that the request limit has never been reached for any of the Crawler processes. As shown in Figure 57, each account has the same performance of the other and the trend of the account shown in Figure 58 exemplifies the trend of the other accounts: each Crawler process consumes at most 100 out of 180 requests every 15 minutes.

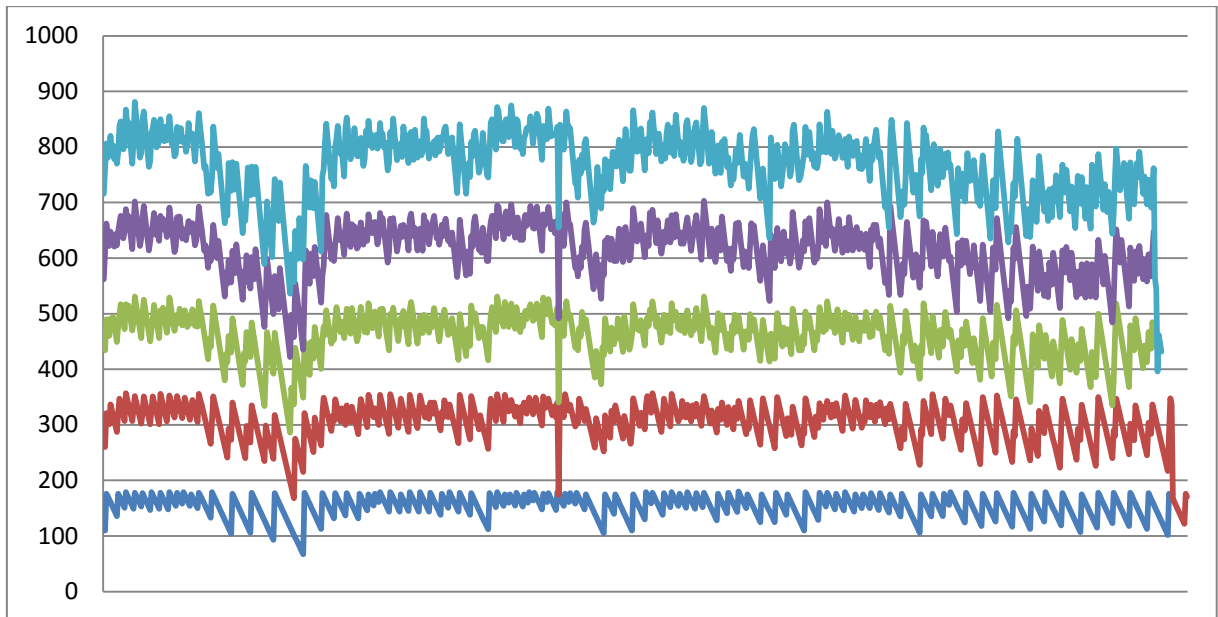


Figure 57: Number of remaining request for each account in a day

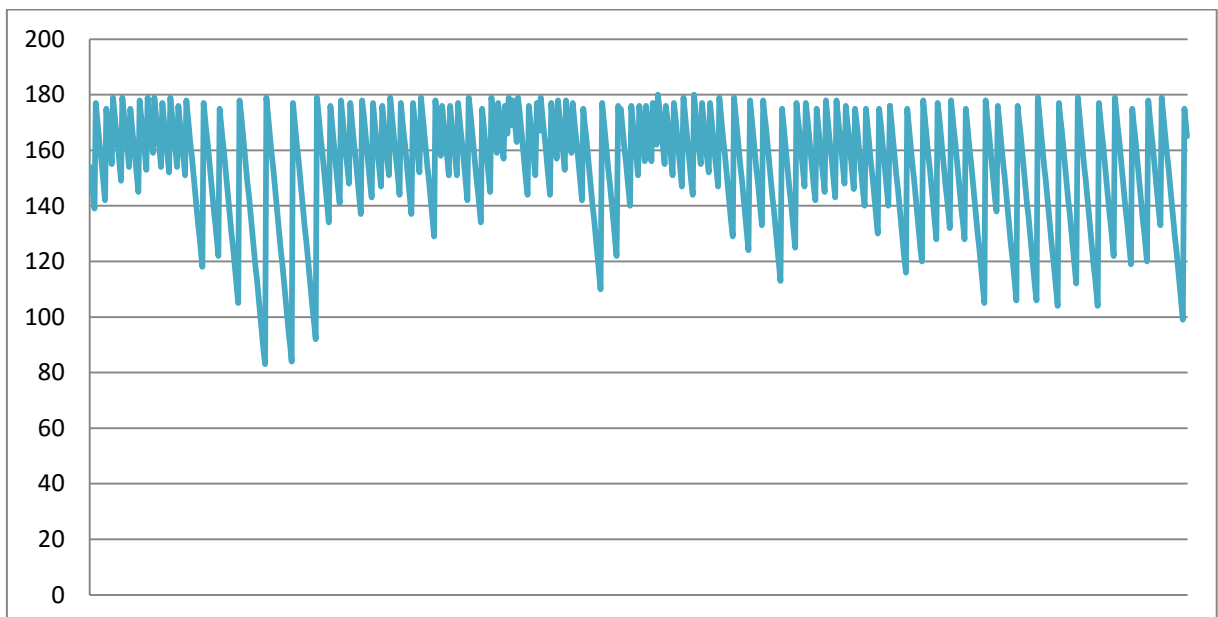


Figure 58: Number of remaining request for an account in a day

Analyzing the percentage of saturation, it can be noted that it is very low, about 3% of the total number of requests made to Twitter. The explanation for this can be given by the delay of adapting to the variation of speed of the search that even if short but affects on this data. In fact, for very fast variations of number of recoverable tweet, for example to the occurrence of a sever weather event, the system reacts already at the first saturation but may be that before achieving stability may occur some saturations.

These results can be summarized by saying that the recovery efficiency of the messages is about 97% and that the recovery efficiency of the tweet fathers is about 99%. Both measures show good results that we have achieved thanks to the designed architecture and to the process recovering fathers tweets. Before the design of this architecture, the platform was based on a process that provided for a single cycle of retrieving of messages: in practice were not made distinctions between searches but were carried out all in a cycle that was repeated after a predetermined time. The efficiency of this process was approximately 85%, with the most active channels that were around 50%. Now the efficiency values of the most active channel are in the order of 95%, while for the other channels never go below 97%.

Conclusions

In this thesis we have been addressed two topics of research that I carried out in the 3-year of PhD.

The first have concerned the study, modelling and validation of an automatic drive system for train in metro rail, the second covered the so-called Social Network Monitoring to use the information flows of Twitter as a way to quantify and model the public attention on certain topics, mainly weather events.

The first research project has dealt with the study, design, development and testing of a system of protection of the train running (Automatic Train Protection, or ATC Automatic Train Control, ATC) of railway vehicles applicable both in the context of interoperable European rail system that in applications of light rail and metro. In particular, the research has focused on protection systems and automatic drive of train in the field of light rail and subway called CBTC (Communication Based Train Control). Starting from the study of standards and products already on the market was decided to model and implement an ATO (Automatic Train Operation) able to be at the forefront over existing systems. To achieve this it was decided to create a system compatible with an ATC system compliant to ERTMS / ETCS standard. In particular, the ATO system has been designed to interface with the ERTMS / ETCS Level 2: to meet this requirement, the system has been implemented to include both the behaviour of the driver interacting with the DMI (Driver Machine Interface) either the behaviour of the DMI. In addition to these requirements the implemented system comply with other specifications such as the initialization of the entire system on board the train, the automatic adjustment of the running of the train, the opening and closing of the doors, etc. In this thesis was presented the process that led to the realization of the Initialization Manager of the train onboard systems: this has been made compatible with ETCS level 2 which deals with the protection of the train running. After the modelling phase, has been exploited the capabilities of IBM Rational Rhapsody tool of automatic code generation to create the source code of the system deriving directly from the model. The behaviour of the model was initially tested through simulation by creating traces of evidence which should simulate messages coming from the ATS and ATC. With these simulations it was possible to verify compliance with the time constraints imposed by the communication protocol DMI-CPU32. After completing the simulation tests, to test the actual operation of the system, various tests were done by integrating the ATO system with the other systems (ATC, ATS, the train interface, etc.).

In the second part has been designed and built a platform for scanning Social Network: for the purpose were used the Twitter API to create a database of messages. This project is a collaboration between UNIFI DISIT laboratory, CNR and LAMMA IBINET to investigate and

build specific metrics and a dashboard reliable to monitor the tweets that refer to the weather. The solution implemented can be used to monitor city services, events and critical circumstances, user behaviour, the city's response to events, etc. Information flows of Twitter have been used as a way to quantify and model the public attention on certain topics. After an initial study of the solutions present on the market, it has been designed an multiprocess system architecture. The main components of this architecture are the Scheduler and the Crawler. The Scheduler is the basic process of the whole system and is a single process that is responsible for starting of Crawler, to update the weights associated to the searches based on the number of new tweets inserted in the database in the last two executions of the research and to update the execution deadline of searches based on the weight assigned previously. The Crawler is responsible for the process to take charge of the search, to forward the requests to Twitter with the selected search parameters and to store the downloaded messages in the database. Can be instantiated multiple Crawler processes and each must be associated with a Twitter account. In addition to the main process for the recovery of the messages have been realized a number of secondary processes useful to the realization of a dashboard for displaying the results of the scan and for the analysis of the messages retrieved. The process for updating the data of the tweet is responsible to recover from Twitter the latest information about messages present in the database. The process for calculating the number of tweets\retweets per day per channel\search takes care of calculating statistics on the number of tweets and retweets divided by channel\search and day. The process for calculating statistics of crawling takes care of calculating the statistics for the Crawling phase: in particular, is calculated the percentage of coverage of father tweets of retweet present in DB and the number of saturations of search. The process for the recovery of missing fathers tweet is responsible for request to Twitter missing messages. To verify the performance of the process of crawling were monitored some values that can reveal the efficiency of the system. The peak of the messages retrieved was 830000 with 5 processes Crawling active. By analyzing the number of requests made by each process can be seen as it has ever been reached the maximum number. These values give reason to believe that the maximum limit in the current configuration of the system is much higher than the limit currently reached. Another parameter that has been analyzed is the number of saturations of the searches. From what was found from the analysis is possible to say that only in the presence of particular events, as an emergency weather, you have some saturations while in most cases saturations not exceed 1% of the total number of requests.

Bibliography

- [1] M. Palumbo, «Railway Signalling since the birth to ERTMS,» 2013.
- [2] «Railway signalling,» [Online]. Available: http://en.wikipedia.org/wiki/Railway_signaling.
- [3] «RusRail,» [Online]. Available: <http://rusrail.net/signaling/stage/>.
- [4] A. Ferrari, «TECHNICAL REPORT CBTC Preliminary Report,» 2011.
- [5] «Railway Technical Web Pages,» [Online]. Available: <http://www.railway-technical.com/atpsurvey.shtml>.
- [6] M. Palumbo, «The ERTMS/ETCS signalling system An overview on the Standard European Interoperable signalling and train control system,» 2014.
- [7] «ERTMS,» [Online]. Available: <http://www.ertms.net/>.
- [8] European Railway Agency, *UNISIG SUBSET-026, ERTMS/ETCS System Requirements Specification, Issue 3.3.0*, 2012.
- [9] R. D. Pascoe e T. N. Eichorn, «What is Communication-Based Train Control?,» *IEEE Veicular Technology Magazine*, 2009.
- [10] Siemens AG, *Trainguard MT The scalable automatic train control system for maximum flexibility in modern mass transit*, 2010.
- [11] J. García, *Sistemas CBTC y automaticos de ansaldo sts*, 2009.
- [12] Institute of Electrical And Electronics Engineers, *IEEE Std 1474.1-2004, IEEE Standard for Communications Based Train Control (CBTC) Performance and Funcotional Requirements*, 2004.
- [13] International Electrotechnical Commission, *IEC 62290-1: Railway applications: Urban guided transport management and command/control systems. Part1: System principles and fundamental concepts*, 2007.
- [14] European Committee for Electrotechnical Standardization, *CENELEC EN 50126, Railway*

applications - The Specification and Demonstration of Reliability, Availability, Maintainability and Safety (RAMS), 2012.

- [15] European Committee for Electrotechnical Standardization, *CENELEC EN 50128, Railway applications - Communications, signalling and processing systems - Software for railway control and protection systems*, 2011.
- [16] European Committee for Electrotechnical Standardization, *CENELEC EN 50129, Communication, signalling and processing systems - Safety related electronic systems for signalling*, 2003.
- [17] A. Ferrari, G. O. Spagnolo, G. Martelli e S. Menabeni, «From Commercial Documents to System Requirements: an Approach for the Engineering of Novel CBTC Solutions,» *International Journal on Software Tools for Technology Transfer (STTT)*, 2013.
- [18] S. Menabeni, G. Martelli, G. Vettori, M. Ignesti e S. Papini, *Applicazione Generica ATO Specifica Preliminare di Sistema*, 2013.
- [19] S. Menabeni, G. Martelli, G. Vettori, M. Ignesti e S. Papini, *CBTC Sistema ATO Specifica dei Requisiti di Sistema*, 2013.
- [20] S. Menabeni e G. Martelli, *Applicazione Generica CBTC Protocollo ATS-ATO*, 2013.
- [21] L. Bellini, «Protocollo di comunicazione DMI ETCS-CPU32,» ECM spa, 2013 [private communication].
- [22] E. Kindler, *Model-based software engineering: the challenges of modelling behavior*, New York, 2010.
- [23] IBM, *IBM Rational Rhapsody - Automatic Test Generation Add On User Guide*.
- [24] M. Faheem e P. Senellart, «Intelligent and Adaptive Crawling of Web Applications for Web Archiving,» in *13th International Conference, ICWE Proceedings*, Aalborg, Denmark, 2013.
- [25] C.-I. Wong, K.-Y. Wong, K.-W. NG, W. Fan e K.-H. Yeung, «Design of a Crawler for Online Social Networks Analysis,» *WSEAS Transactions on Communications*, vol. 13, p. 263, 2014.
- [26] House of Kaizen, «SOCIAL MEDIA MONITORING TOOL BUYER'S GUIDE».
- [27] B. Batrinca e P. C. Treleaven, «Social media analytics: a survey of techniques, tools and platforms,» *AI & SOCIETY*, vol. 30, pp. 89-116, 2015.

- [28] «80legs,» [Online]. Available: <http://80legs.com/>.
- [29] Sequentum, «<http://www.visualwebripper.com/Default.aspx>,» [Online].
- [30] «Helium Scraper,» [Online]. Available: <http://www.heliumscraper.com/en/index.php?p=home>.
- [31] «Prompt Cloud,» [Online]. Available: <http://www.promptcloud.com/>.
- [32] M. Castellanos, M. Hsu, U. Dayal, R. Ghosh, M. Dekhil, C. Ceja, M. Puchi e P. Ruiz, «Intention insider: discovering people's intentions in the social channel,» in *Proceedings of the 15th International Conference on Extending Database Technology*, Berlin, Germany, 2012.
- [33] «Open Amplify,» [Online]. Available: <http://www.openamplify.com/>.
- [34] «Clarabridge,» [Online]. Available: <http://clarabridge.com/>.
- [35] «BrandWatch,» [Online]. Available: <http://www.brandwatch.com/>.
- [36] «Opinion Crawl,» [Online]. Available: <http://www.opinioncrawl.com/>.
- [37] «Social Report,» [Online]. Available: <http://www.socialreport.com/>.
- [38] «Mozenda,» [Online]. Available: <http://www.mozenda.com/>.
- [39] «beevolve,» [Online]. Available: <http://www.beevolve.com/>.
- [40] «Meltwater,» [Online]. Available: <http://www.meltwater.com/>.
- [41] «Viralheat,» [Online]. Available: <https://www.viralheat.com/>.
- [42] «SAS Sentiment Analysis,» [Online]. Available: http://www.sas.com/en_us/software/analytics/sentiment-analysis.html.
- [43] «Dataminr,» [Online]. Available: <https://www.dataminr.com/>.
- [44] «tracx,» [Online]. Available: <http://www.tracx.com/>.
- [45] «Royalty,» [Online]. Available: <http://www.roialty.com/>.
- [46] A. Talamo, *Progettazione e sviluppo di un modulo per l'integrazione di Twitter nel CMS Drupal e analisi descrittiva del flusso dei Tweets*, Firenze, 2015.
- [47] «Twocharts, sito che fornisce statistiche ufficiali su twitter,» [Online]. Available:

<http://twopcharts.com>.

[48] «Supporto Twitter,» [Online]. Available: <https://support.twitter.com/articles/119138-types-of-tweets-and-where-they-appear>.

[49] F. Flamini, «Sistemi di controllo per l'Alta Velocità ferroviaria,» *Mondo Digitale*, pp. 18-24, 2010.

Table of Figures

Figure 1: Fixed-block signaling system	9
Figure 2: Safety distance between trains in fixed block and moving block signalling systems	10
Figure 3: European ATC/ATC systems before ERTMS/ETCS.....	12
Figure 4: Level 1.....	13
Figure 5: Level 2.....	14
Figure 6: Level 3.....	15
Figure 7: Service Speed Profile and Signaling Speed Profile	16
Figure 8: GoA Levels	19
Figure 9: ATO Operating Context	24
Figure 10: ATO Generic Application Architecture	25
Figure 11: CBTC System Architecture	31
Figure 12: Components involved in Initialization	32
Figure 13: Preliminary operation of Initialization phase.....	33
Figure 14: Flowchart for Start of Mission Procedure	41
Figure 15: Init block definition diagram	46
Figure 16: Statechart of Initialization_Manager	47
Figure 17: Sub-states of Preliminary operations state.....	48
Figure 18: Sub-states of Preliminary_operations state.....	49
Figure 19: Sub-states of Management_DriverID state.....	49
Figure 20: Sub-states of Management_Level state.....	50
Figure 21: Sub-states of Management_RBCCont state.....	50
Figure 22: Sub-states of Waiting_EnterData state.....	51
Figure 23: Sub-states of Management_TrainData state	51
Figure 24: Sub-states of Management_TrainDataVal state	52
Figure 25: Sub-states of Management_TRNN state.....	52
Figure 26: Sub-states of Waiting_start_eneabled state.....	53
Figure 27: Sub-states of Ready state	53
Figure 28: Sub-states of Management_Idle state	54
Figure 29: Wait_for_Termination state.....	54
Figure 30: Testing Methodogy Workflow	56
Figure 31: JSON example	73
Figure 32: Crawling system architecture	86
Figure 33: Database architecture	87
Figure 34: Class Diagram for Crawler process	99
Figure 35: State diagram for TwitterCrawler.....	100
Figure 36: Class diagram for TwitterScheduler	101
Figure 37: State diagram for TwitterScheduler	102
Figure 38: State diagram for ranking update (<i>rank</i> is the priority associated with Search, <i>media</i> is the average of the results of the last 2 execution of Search)	104

Figure 39: Structure of User Frontend	108
Figure 40: Channel statistics page	109
Figure 41: Statistics on single Channel page	110
Figure 42: Search statistics page	111
Figure 43: Search statistics page: single channel details.....	112
Figure 44: Twitter Users statistics	113
Figure 45: Details of the number of users grouped by search	114
Figure 46: Details of user grouped by Search.....	115
Figure 47: User profile	115
Figure 48: Retweet statistics page.....	116
Figure 49: Retweet statistics for single Channel page	117
Figure 50: Structure of Administrator Frontend	118
Figure 51: Search parameters page.....	119
Figure 52: Search parameters page -> add new channel.....	120
Figure 53: Add new Search form	120
Figure 54: Crawler statistics page.....	121
Figure 55: Processes status page.....	122
Figure 56: Number of Tweets downloaded per day.....	124
Figure 57: Number of remaining request for each account in a day	125
Figure 58: Number of remaining request for an account in a day	125

List of Tables

Table 1: Requirement Qualifiers.....	27
Table 2: Start of Mission procedure	40
Table 3: Operators for Twitter query	69
Table 4: JSON fields	81
Table 5: account_twitter table	88
Table 6: ax_request_twitter table	89
Table 7: ax_twitter page.....	90
Table 8: ax_twitter_geocode table.....	90
Table 9: canale table.....	91
Table 10: chart_eventi_canali table	91
Table 11: chart_twitter table.....	92
Table 12: chart_twitter_canali table	92
Table 13: chart_twitter_retweet table.....	93
Table 14: chart_twitter_retweet_canali table	93
Table 15: chart_user table.....	93
Table 16: chart_user_canali table	94
Table 17: crawling_stat table	95
Table 18: crawling_stat_day table.....	95
Table 19: hashtags table.....	96
Table 20: mentions table.....	96
Table 21: process_list table	96
Table 22: tweet_retrieve table.....	97
Table 23: users table.....	97
Table 24: variable table	97
Table 25: Crawling statistics	123