REVEAL
FP7-610928

REVEALing hidden concepts in Social Media

Deliverable D6.1.2
Architecture and specifications

Editor(s): Efstratos Tzoannos, Nikos Sarris
Responsible Partner: ATC
Status-Version: Final – v1.0
Date: 15/05/2015
EC Distribution: Public

Project Number: FP7-610928
Project Title: REVEAL
<table>
<thead>
<tr>
<th>Title of Deliverable:</th>
<th>Architecture and Specifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Date of Delivery to the EC:</td>
<td>15/05/2015</td>
</tr>
<tr>
<td>Workpackage responsible for the Deliverable:</td>
<td>WP6 – Framework Integration and REVEAL Applications</td>
</tr>
<tr>
<td>Editor(s):</td>
<td>Efstratios Tzoannos, Nikos Sarris</td>
</tr>
<tr>
<td>Contributor(s):</td>
<td>NCSR’D’, CERTH, UKOB, ITINNO, ALBLF</td>
</tr>
<tr>
<td>Reviewer(s):</td>
<td>NCSR’D’</td>
</tr>
<tr>
<td>Approved by:</td>
<td>All Partners</td>
</tr>
</tbody>
</table>

**Abstract:**

This document details the design of the core framework of the platform as well as the general design of the system modules. This is an updated version of D6.1.1 including all the developments that have been decided up to this point.

**Keyword List:**

model, architecture, specifications, testing, software, storm, integration, presentation
# Document Revision History

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Modifications Introduced</th>
<th>Modified by</th>
</tr>
</thead>
<tbody>
<tr>
<td>v0.1</td>
<td>23/04/2015</td>
<td>ToC, and a first draft</td>
<td>ATC</td>
</tr>
<tr>
<td>V0.2</td>
<td>27/04/2015</td>
<td>Included and updated content from D6.1.1</td>
<td>ATC</td>
</tr>
<tr>
<td>V0.3</td>
<td>13/05/2015</td>
<td>Updated content based on discussions in the plenary meeting and information received from the module developing partners</td>
<td>ATC</td>
</tr>
<tr>
<td>V1.0</td>
<td>15/05/2015</td>
<td>Updated content based on review by NCSR</td>
<td>ATC</td>
</tr>
</tbody>
</table>
## CONTENTS

**EXECUTIVE SUMMARY** ................................................................. 10

1 **INTRODUCTION** ........................................................................ 11
   1.1 PURPOSE OF THIS DOCUMENT .................................................. 11
   1.2 DOCUMENT STRUCTURE ............................................................ 11

2 **ARCHITECTURE OVERVIEW** .................................................... 12
   2.1 ARCHITECTURAL PRINCIPLES .................................................. 12
   2.2 LAYERED ARCHITECTURE PATTERN ......................................... 12
   2.3 DISTRIBUTED REALTIME COMputation WITH STORM ............... 14
   2.4 OVERALL ARCHITECTURE OF REVEAL PLATFORM ..................... 15

3 **DOMAIN DRIVEN DESIGN** ....................................................... 17
   3.1 CORE DOMAIN AND DOMAIN LOGIC ......................................... 17
   3.2 UBQUITOUS LANGUAGE ......................................................... 17

4 **INTEGRATION** ........................................................................... 19
   4.1 AGILE METHODOLOGY ............................................................. 19
   4.2 INTEGRATION WITH DOCKER .................................................. 20
      4.2.1 Docker Guidelines ............................................................. 21
   4.3 DOCKER PRIVATE REGISTRY .................................................... 23
   4.4 CONTINUOUS INTEGRATION ................................................... 24
   4.5 IMPLEMENTATION PROCESS .................................................. 30
   4.6 THE TICKET WORKFLOW ........................................................ 30
   4.7 HOW TO DEFINE TICKET PRIORITY ........................................... 30
   4.8 RELEASE MANAGEMENT ......................................................... 30
   4.9 REVEAL INTEGRATION ROADMAP .......................................... 31

5 **PRESENTATION LAYER** ............................................................ 33
   5.1 INTERFACES ............................................................................. 33
   5.2 TECHNOLOGIES/DEPENDENCIES ........................................... 33
      5.2.1 AngularJS ........................................................................ 33
   5.2.2 jQuery ............................................................................... 33
   5.2.3 Bootstrap .......................................................................... 34
   5.2.4 HTML5/CSS ....................................................................... 34
   5.2.5 Decision support visualisation (ITINNO) .................................. 34

6 **BUSINESS LAYER** ..................................................................... 35
   6.1 CRAWLING AND INDEXING .................................................... 35
      6.1.1 ITINNO crawler ................................................................. 35
   6.1.2 Focused Image Crawler (Web Crawler) .................................... 35
6.1.2 WP2 crawler – Twitter RESTful API Crawler ............................................................. 36
6.1.4 SAG crawler (services to fetch data) ..................................................................... 37
6.1.5 RabbitMQ2Mongo .................................................................................................. 38

6.2 ORCHESTRATION .................................................................................................. 39
6.2.1 Offline Process Orchestrator (asynchronous) ...................................................... 39
6.2.2 Offline Process Orchestrator (synchronous) ....................................................... 41
6.2.3 Storm Orchestrator .............................................................................................. 41
6.2.4 Communication .................................................................................................... 42

6.3 SECONDARY MODULES ...................................................................................... 42

6.4 ANALYSIS MODULES (MODULE ID CARD) ...................................................... 44
6.4.1 Multimedia clusterer (CERTH) ............................................................................ 44
6.4.2 Multimedia geoleocator (CERTH) ....................................................................... 44
6.4.3 Multimedia indexing (CERTH) ........................................................................... 44
6.4.4 Multimedia similarity retriever (CERTH) ........................................................... 44
6.4.5 Multimedia Summariser (CERTH) ...................................................................... 45
6.4.6 Multimedia manipulation detector (CERTH) ..................................................... 45
6.4.7 Fake Post Detector (CERTH) .............................................................................. 45
6.4.8 Multimedia social context builder and modality extractor (CERTH) .................. 45
6.4.9 User network profile classifier (CERTH) ............................................................. 46
6.4.10 Web crawler (CERTH) ....................................................................................... 46
6.4.11 Community ranking (CERTH) .......................................................................... 47
6.4.12 Geospatial context extraction (pre-) processing (ITINNO) ............................... 47
6.4.13 Situation assessment framework (ITINNO) ....................................................... 47
6.4.14 Social media client framework (ITINNO) .......................................................... 48
6.4.15 Focused Geospatial exploration (UniKob) .......................................................... 48
6.4.16 Geospatial topic model (UniKob) ....................................................................... 48
6.4.17 Role analysis (UniKob) ...................................................................................... 49
6.4.18 Semantic context model (UniKob) ..................................................................... 49
6.4.19 Social context model (UniKob) ......................................................................... 50
6.4.20 Topical context model (UniKob) ....................................................................... 50
6.4.21 Source finding (Alblf) ....................................................................................... 51
6.4.22 Influence calculation (Alblf) .............................................................................. 51
6.4.23 Contributor alignment (Alblf) ............................................................................. 51
6.4.24 Multpartite Community detection (NCSR) ....................................................... 52

Project Title: REVEAL
Project Coordinator: INTRASOF T International S.A. 
Contract No. FP7-610928
www.revealproject.eu
6.4.25 Influence calculation (NCSR) ................................................................. 52
6.4.26 Pserver (NCSR) ...................................................................................... 53
6.4.27 Relation discovery (NCSR) ..................................................................... 53
6.4.28 Semantic segmentation (NCSR) ............................................................. 54
6.4.29 Stylometry (NCSR) ................................................................................ 54
6.4.30 Presence (NCSR) ................................................................................... 54
6.4.31 Community Tracking (NCSR) ............................................................ 55
6.4.32 Complex Event Recognition (NCSR) .................................................. 55
6.4.33 Community Evolution Prediction (NCSR) .......................................... 56
6.4.34 Linguistic analysis (NSCR) .................................................................. 57
6.4.35 Trust and credibility model (Ukob-ALCATEL) ..................................... 57
6.4.36 Trust and credibility model (ITINNO) .................................................. 58
6.4.37 Sentiment Analysis (ATC) ........................................................................ 58

7 DATA ACCESS - PERSISTENCE LAYER ............................................. 59

7.1 DATA ACCESS OBJECTS ........................................................................... 59
7.2 APIs – PYTHON/JAVA FOR MONGODB .................................................. 59
  7.2.1 Java wrapper API for MongoDB ............................................................ 59
  7.2.2 Python wrapper API for MongoDB ....................................................... 62
7.3 STORM RABBITMQ DATA ACCESS (JAVA/JAVAPYTHON RABBITMQ CLIENT) .................. 64
7.4 APACHE SOLR DATA ACCESS ................................................................. 64
7.5 COMMON DATA TRANSFER OBJECTS (WIKI) ...................................... 64
7.6 REPOSITORIES ........................................................................................... 65
  7.6.1 MongoDB ............................................................................................ 65
  7.6.2 Apache Solr ........................................................................................ 65

8 PHYSICAL DEPLOYMENT ......................................................................... 66

8.1 HARDWARE SPECIFICATIONS ................................................................. 66
8.2 INFRASTRUCTURE ...................................................................................... 67

9 TESTING ........................................................................................................ 68

10 CONCLUSION ............................................................................................... 70

11 APPENDIX A: REVEAL VOCABULARY .............................................. 71

12 APPENDIX B: BASIC POST OBJECT (ITEM) AS RETRIEVED BY THE ITINNO CRAWLER. .................................................................................................. 73

13 APPENDIX C: BASIC POST OBJECT (ITEM) AS EXPOSED BY SAG ARIS COMMUNITY SERVICES .................................................................................. 74

14 APPENDIX D: USER INFO OBJECT AS EXPOSED BY SAG ARIS COMMUNITY SERVICES .................................................................................................. 75
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Layered Architecture</td>
<td>13</td>
</tr>
<tr>
<td>2</td>
<td>Overall Architecture</td>
<td>15</td>
</tr>
<tr>
<td>3</td>
<td>REVEAL Continuous Integration</td>
<td>25</td>
</tr>
<tr>
<td>4</td>
<td>REVEAL Projects in Jenkins</td>
<td>26</td>
</tr>
<tr>
<td>5</td>
<td>REVEAL Gantt Diagram from Redmine</td>
<td>27</td>
</tr>
<tr>
<td>6</td>
<td>REVEAL Redmine Community</td>
<td>27</td>
</tr>
<tr>
<td>7</td>
<td>REVEAL Projects in GitLab</td>
<td>28</td>
</tr>
<tr>
<td>8</td>
<td>REVEAL Development Community</td>
<td>28</td>
</tr>
<tr>
<td>9</td>
<td>JMongoHandler in Sonatype Nexus</td>
<td>29</td>
</tr>
<tr>
<td>10</td>
<td>Repository Definition in POM File</td>
<td>30</td>
</tr>
<tr>
<td>11</td>
<td>Offline Process Orchestrator</td>
<td>40</td>
</tr>
<tr>
<td>12</td>
<td>Sequence Diagram for Offline Orchestration</td>
<td>41</td>
</tr>
<tr>
<td>13</td>
<td>REVEAL Python Mongo Handler Description</td>
<td>63</td>
</tr>
<tr>
<td>14</td>
<td>The ISO/IEC 25010:2011 System/Software Quality Model Characteristics</td>
<td>69</td>
</tr>
</tbody>
</table>
LIST OF TABLES

TABLE 1: REVEAL VOCABULARY .................................................................72
## DEFINITIONS, ACRONYMS AND ABBREVIATIONS

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>ASD</td>
<td>Agile Software Development</td>
</tr>
<tr>
<td>CI</td>
<td>Continuous integration</td>
</tr>
<tr>
<td>DB</td>
<td>Database</td>
</tr>
<tr>
<td>DDD</td>
<td>Domain Driven Design</td>
</tr>
<tr>
<td>GUI</td>
<td>Graphical User Interface</td>
</tr>
<tr>
<td>HTTP</td>
<td>Hypertext Transfer Protocol</td>
</tr>
<tr>
<td>IP</td>
<td>Internet Protocol</td>
</tr>
<tr>
<td>POM</td>
<td>Project Object Model</td>
</tr>
<tr>
<td>R&amp;D</td>
<td>Research and Development</td>
</tr>
<tr>
<td>REST</td>
<td>Representational State Transfer</td>
</tr>
<tr>
<td>SDK</td>
<td>Software Development Kit</td>
</tr>
<tr>
<td>SOA</td>
<td>Service Oriented Architecture</td>
</tr>
<tr>
<td>UI</td>
<td>User Interface</td>
</tr>
<tr>
<td>URL</td>
<td>Uniform Resource Locator</td>
</tr>
</tbody>
</table>
Executive Summary

This document is an updated version of D6.1.1. It details the latest results of the system design process and the conceptual architecture design, which was developed accordingly.

The adopted system architecture was based on iterative discussions, refactoring, and an enrichment process involving all REVEAL module developers and technology providers that are part of the consortium.

Following the requirement analysis and the study of a number of related systems, an initial architecture is presented that was derived according to established design principles and in direct correspondence with the identified system requirements. The architecture specifies the main functional components of the envisioned system and the foreseen methods for their communication and collaboration.
1 Introduction

1.1 Purpose of this document

The objective of this document is to present the overall architecture of the REVEAL platform. The current document outlines the platform architecture including a general description of the components layout, their internal architecture as well as the interactions among them. The flows and architecture of the REVEAL platform is defined in terms of the supported functionalities, the respective processes and the components that realise them.

This will serve as a reference point for the development work that will take place in the technical work-packages (WP2-WP5). The decisions presented in this deliverable are subject to refinements and modifications, based on the progress of the technical work packages, as well as the validation and evaluation phases.

1.2 Document structure

The following sections in this deliverable are:

Section 2, where the REVEAL architecture overview is presented, based on the user requirements elicitation process and resulting technical requirements.

Section 3 describes the domain driven design approach in relation to REVEAL.

Section 4 presents an overview of the integration strategy that will be followed, together with a timeline for following up on the related activities.

Section 5 lists all the technologies and modules that contribute to the realization of the front-end of the REVEAL platform.

Section 6 lists all the modules that make up the REVEAL business layer. Further, it describes the orchestration approach followed in order to fulfil the user scenarios.

Section 7 describes how data is stored and retrieved and the technologies used.

Section 8 presents the envisioned physical deployment of the REVEAL platform.

Section 9 outlines the testing procedure that will be followed on the functionalities of each module and of the platform as a whole to ensure the quality of REVEAL’s implementation.

Finally, Section 10 concludes the deliverable.
2 Architecture overview

2.1 Architectural principles

We consider a design as successful when it covers all the following aspects:

- Usability
- Performance
- Security
- Maintainability
- Scalability

In the architectural design of the REVEAL platform we are addressing all of the above in order to make a robust and fully flexible system, which will cover all the needs of REVEAL project. From a conceptual point of view, as most modern software architectures, it follows a layered architecture pattern. Thus, the system’s modules are divided into four distinct layers, each responsible for a specific set of functionalities. By convention, the layers interact with each other in a top-down manner, with each layer being able to access all layers below. A lower layer should never interact with layers above. This convention helps to avoid circular dependencies between layers.

2.2 Layered Architecture Pattern

The layers in a generic layered architectural design of any software platform are described as follows:

- **Presentation Layer:** Contains all the User Interfaces and Visualization Modules
- **Services Layer:** Exposes multiple APIs in the form of web services, defining a set of resources/methods as well as message structures
- **Business Layer:** Encapsulates all the business logic, as well as core domain entities of the system. It implements all system’s workflows and offers a simplified API (system façade) to the top layers for fulfilling the business workflows.
- **Data Layer:** Consists of all Data Access Objects as well as external service consumers. It is the broker to all the persistence storage and external data.
- **Cross-Cutting Layer:** Although it is not one of the basic four layers, it contains a set of features and modules which do not belong to a specific layer, since they are collaborating with all layers of the platform. These modules refer mainly to security and communication.

The following diagram¹ depicts the layered software architecture:

The layered architectural design addresses all the aspects that we are targeting. Together with other software architectures and standards that are followed, it helps us to apply the best standards in all the design aspects we are focusing on. More specifically:

**Usability**: The fact that we are following a layered architecture isolates the presentation modules from the logic layer, giving the possibility to focus on good User Experience design (UX). Thus, a web designer or a usability expert can work separately on the User Interface unaffected by the backend system developers. These experts can focus distinctly on the User Interface to maximize the quality of user experience. Internally, inside the presentation layer, we are going to follow a Model View Controller design pattern for building a web application, starting from a plain, well-defined user interface that consumes the services provided by the backend. The use of cutting-edge technologies like HTML5, CSS3 and jQuery will offer the best set of front-end features to provide a clean and fully functional interface. Finally, we are going to follow an agile methodology for developing the platform, based on rapid prototyping and frequent iterations. This enables more frequent evaluations close to the end-user of the platform and better result in terms of meeting the usability requirements.

**Performance**: For tackling performance issues, we are going to rely on two factors – caching and distribution. The system architecture logic is implemented in the backend and is exposed through an API of RESTful web services. These services offer parallelization in calls to the backend and can be deployed independently in a distributed way among several nodes of the cloud, following a Software Service Oriented Architecture. If needed, we could also use an Enterprise Service Bus to merge several heterogeneous services of the backend. If needed, load balancing and caching will be
applied between the presentation and the services layers. Additionally, the caching of data can take place even between the business layer and the data layer, when frequent fetching of the same data is required.

**Security:** The security features will be applied system-wise, covering all layers of architecture. The external API (Service Interfaces) exposed by the platform can be secured by means of encryption over HTTP via SSL, which is the standard protocol for security over the Internet. Only the Services Layer and the UI is exposed publicly. Therefore, for the rest of the system (e.g. Databases) we don’t need further security since it is located inside a closed environment, which will be secured from the outside world by means of the cloud’s firewall. Thus, no one will be able to access the business or data layer but only the modules that reside on the same LAN.

**Maintainability:** For addressing maintainability, we are going to follow standards oriented and technology independent architecture. Focusing on well-defined generic standards (e.g REST) we don’t rely on specific knowledge for proprietary solutions and standards. We will use only open-source solutions written in JVM languages (Java, Groovy) or Python and Javascript that are cross-platform and fully flexible. For code collaboration tools, we are going to use an overall Continuous Integration solution with Hudson/Jenkins, maven, sonar and subversion for code versioning management. For communication between the various developer teams, we are going to use a Redmine ticket tracking system. Trac is an interesting alternative solution that provides similar functionality. This way all tickets can be traceable and all code commits can be examined and explained, referring to specific tickets.

**Scalability:** The system is going to be able to scale up based on the volume of the requests. The elasticity will be provided by the cloud platform itself by configuring the resources appropriately. The architecture is free to scale up easily due to the service oriented distributed nature of the backend.

## 2.3 Distributed Realtime Computation with Storm

Apache Storm\(^2\) is a free and open source distributed real-time computation system. Storm makes it easy to reliably process unbounded streams of data, doing for real-time processing what Hadoop did for batch processing. Storm is simple, can be used with any programming language. Storm has many use cases: real-time analytics, online machine learning, continuous computation, distributed RPC, ETL, and more. Storm is fast: a benchmark clocked it at over a million tuples processed per second per node. It is also scalable and fault-tolerant.

Storm distinguishes between the following three main entities that are used to actually run a topology in a Storm cluster:

- Worker processes
- Executors (threads)
- Tasks

A worker process executes a subset of a topology, and runs in its own JVM. A worker process belongs to a specific topology and may run one or more executors for one or more components (spouts or bolts) of this topology. A running topology consists of many such processes running on many machines within a Storm cluster.

An executor is a thread that is spawned by a worker process and runs within the worker’s JVM. An executor may run one or more tasks for the same component (spout or bolt). An executor always has one thread that it uses for all of its tasks, which means that tasks run serially on an executor.

---

\(^2\) Apache Storm: [https://storm.incubator.apache.org/](https://storm.incubator.apache.org/)
A task performs the actual data processing and is run within its parent executor’s thread of execution. Each spout or bolt that you implement in your code executes as many tasks across the cluster. The number of tasks for a component is always the same throughout the lifetime of a topology, but the number of executors (threads) for a component can change over time.

Storm is a key part of the REVEAL ecosystem consisting of a single storm controller per partner and many storm topologies. WP6 pilot applications will be invoking the Storm controllers through an HTTP endpoint for initiating new processing procedures. Each storm topology is a packaged JAR file containing a storm runner class (Java) to launch the topology and several storm bolt processing classes (Java, Python) to perform the processing acts required.

## 2.4 Overall Architecture of REVEAL Platform

The overall architecture of REVEAL is depicted in the following diagram:

![Overall Architecture Diagram](image)

We could distinguish the components of this diagram in the following groups:

### User Interfaces (UI)

This is the entry point for every user who has access to the platform. As seen in this diagram, all actions start from the pilot application (UI). From the UI, a user can start a new campaign which invokes a new crawling activity, can start the analysis workflows of the backend and can also visualise the results of the analysis. The UI consists of the news app together with the Decision Support Visualisation plugins. There is also a UI covering the Enterprise use case.
Crawlers

The group of crawlers is responsible for fetching content from the Internet to our platform. A crawler can start either manually by a user during the initiation of an assessment, or it is automatically invoked by the system itself (e.g. in an all-purpose crawl to fetch generic content).

The main crawler is Social Media Client Framework module, which populates the RabbitMQ message broker with new content. This content is also copied inside a mongoDB NoSQL database (through MQ2mongo module). There will be a continuous generic crawling activity with default specifications and there will also be focused crawling activity invoked through specific requests from the UI.

A second crawler is the web crawler, which is fetching multimedia and news content from the Internet. Finally, a new crawler implemented for the needs of WP2 is the Twitter RESTful API crawler, which is performing a generic crawl based on the Twitter's RESTful API. This can be considered part of the Social Media Client Framework, therefore, no new module is inserted in the diagram.

Analysis Modules and Controlling Modules

These modules form the business layer of the platform. They are responsible for the implementation of the analysis logic of the collected content. The backend analysis of the fetched raw content is divided in offline (batch) process and online (storm) process. For triggering new storm topologies, the UI uses the storm controllers, which expose a RESTful HTTP interface. The design pattern for Storm in REVEAL is to create a single storm controller (per partner) and many storm topologies. Each storm controller is a HTTP endpoint that allows pilot applications to easily start new processing for new situation assessments. Each storm topology is a packaged JAR file containing a storm runner class (Java) to launch the topology and several storm bolt processing classes (Java, Python) to perform the processing acts required (e.g. geo-parsing). The input for each storm topology and the communication between the storm modules is performed through the RabbitMQ message broker.

For the offline process, the orchestrator is responsible for controlling the invocation and termination of the various offline modules. For this purpose, all modules are built as executable files (or Docker containers), which can be easily triggered by the orchestrator. Since we are following a distributed deployment, the modules are sending notifications about their status to a central RabbitMQ message broker. A more detailed description about the offline analysis orchestrator is provided in subsequent section of this document.

Persistent Layer - Databases

As already mentioned, the core DBs of the system consist of a RabbitMQ message broker and NoSQL mongodb database. The RabbitMQ message broker is used as communication channel and buffer for pushing content fetched by the crawlers. The mongodb database is used for permanent storage, mainly used for batch processing by the offline analysis. The persistent layer includes also secondary databases, used internally for specific purposes in some or all topologies/processes (Berkley DB, Apache Solr, neo4j, RDF triple store etc.). The detailed description of each module, as well as the orchestration/communication mechanisms to be used in every layer is described in the following sections.

---

3 REVEAL D5.1: Analytic report of metadata template definitions
3 Domain Driven Design

3.1 Core domain and domain logic

Domain-Driven Design (DDD)\(^4\) is an approach to software development for complex needs by connecting the implementation to an evolving model. In domain-driven design, three basic uses determine the choice of a model.

- The model and the design of the product are tightly bound. This binding of model and implementation also helps during maintenance and continuing development, because the code can be interpreted based on understanding the model.
- The model is the backbone of a language used by all team members
- The model is an agreed-upon way of structuring key elements in a project to facilitate partner collaboration

This model-based communication is not limited to diagrams in Unified Modelling Language (UML). To make most effective use of a model, it needs to pervade every medium of communication. It increases the utility of written text documents, as well as the informal diagrams and casual conversation reemphasized in Agile processes. It improves communication through the code itself and through the tests for that code.

3.2 Ubiquitous Language

A project faces serious problem when its language is fractured. Usually, domain experts have limited understanding of the technical terms of software development, but they use the terms of their field. Developers, on the other hand, may understand and discuss the system in descriptive, functional terms, devoid of the meaning carried by the experts’ language. Alternatively, developers may create abstractions that support their design but are not understood by the domain experts. Developers working on different parts of the problem work out their own design concepts and ways of describing the domain.

The terminology of day-to-day discussions is disconnected from the terminology embedded in the code (ultimately the most important product of a software project). On a project without a common language, developers have to translate for domain experts. Also, domain experts translate between developers and still other domain experts. Developers even translate for each other. Translation frequently hides model concepts, which leads to inefficient refactoring of code. The overhead cost of all the translation is reflected in the quality of the product.

A project needs a common language that is more robust and offers a channel of communication for every developer or domain expert. This implies continuous effort by all teams for the formulation of a common language, which derives directly from the module of the system. This language will be used for the communication of all members of for the software implementation.

The vocabulary of that Ubiquitous Language includes the names of classes and prominent operations. The language includes terms to discuss rules that have been made explicit in the model. This language is also enriched with the names of patterns the team commonly applies to the domain model.

The model-based language should be used among developers to describe not only artifacts in the system, but tasks and functionality. This same model should supply the language for the developers and domain experts to communicate with each other, and for the domain experts to communicate among themselves about requirements, development planning and features.

With a Ubiquitous Language, conversations among developers, discussions among domain experts, and expressions in the code itself are all based on the same language, derived from a shared domain model.

In the scope of REVEAL, we have started formulating such a ubiquitous language to be used among all the teams through a collaborative process between various team members (both domain experts and technical experts). This has resulted in a live document persisted in the wiki of the project, which is expected to evolve throughout the lifecycle of REVEAL project. A current snapshot of this document is attached in the appendix of this deliverable.
4 Integration

For the integration purposes of REVEAL project we are going to follow Agile Software Development Practices with frequent integration cycles, rapid prototyping and close collaboration between self-organising, cross-functional teams (section 4.1). Based on agile principles, we are also going to apply Continuous Integration techniques for performing automated building, testing and deployment of the provided modules. For adopting Continuous Integration practices we are going to setup a development environment containing a set of continuous integration tools. The tools and the workflow of the continuous integration process is described in section 4.2 of this document. The modules that will take part in the continuous integration workflow shall be based on maven.

Except for the continuous integration approach, we are going to follow a remote deployment setup by the developers of fully functional Docker containers, which are going to be delivered to the integration team. This approach will be mainly used for the setup of the various Storm environments (3rd party dependencies + repositories), which are going to host the different Storm topologies. Once a Docker container is configured, it will be moved to the production server, which will contain the main Docker installation. Further information about the setup of Strom clusters can be found in section 4.3 of this document.

4.1 Agile Methodology

Agile software development (ASD)\(^5\) is a group of software development methodologies based on iterative and incremental development, where requirements and solutions evolve through collaboration between self-organising, cross-functional teams, addressing the development efforts performed in the various stage of a project.

There are many specific ASD methods. Most of them promote development, teamwork, collaboration, and process adaptability throughout the life-cycle of the project. ASD goes beyond traditional software development processes (such as Waterfall) and exploits an evolutionary method that is an iterative and incremental approach to software development and integration. Thus, the requirements and design phases are iteratively met with the development phase to incrementally produce system software releases, which can be assessed over the suitability, the maturity and the immediate business value. On top of them, ASD foresees an intense testing phase, in which the unit testing is achieved from the developer’s perspective and the acceptance testing is conducted from the customer’s perspective. Thus, the major difference with respect to the iterative approaches of plan-based methodologies (like RUP and Spiral) is the fact that requirements and testing are part of the actual development iterative process and the target stakeholders can be progressively involved in the development process aiming to deliver high quality software.

“The best architectures, requirements, and designs emerge from self-organising teams”. ASD manifests that “working software is the primary measure of progress”. Thus, agile methodology approaches the requirements of a software environment, like the cloud market, iteratively by frequently delivering working software prototypes. These prototypes enable the project development and business teams to work together and maximise the quality of the produced output.

ASD can be suitable for the development strategy of an ICT solution, mainly because:

\(^5\) [http://agilemethodology.org/](http://agilemethodology.org/)

---

Project Title: REVEAL  
Contract No. FP7-610928  
Project Coordinator: INTRASOFInternational S.A.  
www.revealproject.eu
• A parallel process between the development of the planned software solution and the verification of the requirements can be followed, leading to business oriented people to actively participate in the specification of use cases and the evaluation of the system developments and provide valuable feedback in an iterative way;
• The work on individual and independent development fields is split among small groups comprising the separate development teams;
• As a ready to the market solution is envisaged rapidly, the solution can benefit from frequent releases to align the work done among the individual teams;
• The producing releases can be exchanged among senior technical teams and business oriented groups to evaluate the effectiveness of the solution in real business situations.

Extreme Programming (XP) and SCRUM are the most popular agile-based software development methodologies. Such methodologies can decrease the time required to produce releases and engage the customer in the development process, maximising the possibilities for a high quality output. On the other hand, the effectiveness of these methodologies is tightly coupled to the proper communication between business end users and engineering teams, while the lack of specific documentation might put the final acceptance at risk.

4.2 Integration with Docker

For those components that will be delivered as Storm topologies or involve complex environment setup there is the need to ensure that the hosting environment (i.e. a Storm cluster) has been set up appropriately and that all software dependencies are met. A Storm cluster installation is not a trivial procedure and consists of the installation and configuration of Storm and Zookeeper as well as the running of their respective daemons under process supervision, in a number of nodes. The idea is to take advantage of Docker and be able to reproduce the Storm cluster stack as a unit of software deployment that can be shared by the REVEAL partners, ensuring thus that is configured as expected.

For these purposes we have created a private Docker registry to share with all the partners that are going to provide their modules as Docker images.

Docker is an open platform for developers and sysadmins to build, ship, and run distributed applications. Consisting of Docker Engine, a portable, lightweight runtime and packaging tool, and Docker Hub, a cloud service for sharing applications and automating workflows, Docker enables apps to be quickly assembled from components and eliminates the friction between development, QA, and production environments. ⁶

We have installed Docker platform v.1.3 at reveal.atc.gr server and we have released the following guidelines to the Reveal Development Community. We have already started committing the first module images successfully.

The following diagram explains the mechanism of integration with Docker. More specifically, we are going to install the Docker platform to a set of Ubuntu servers. Then, when a partner is ready to release an all-in-one Docker image, this image is going to be committed to the REVEAL private Docker registry. After it is uploaded to the registry, the integration team will retrieve and run it as a Docker container to one (or multiple) Docker platforms. After this deployment takes place, for retrieving subsequent versions of the same modules we have two options: a) either the partners re-commits the updated Docker image to the REVEAL private Docker registry or b) the partner submits

⁶ Docker - An open platform for distributed applications for developers and sysadmins. https://www.docker.com/whatisdocker/
only the sub-modules to be integrated (e.g. a web app which has changed) instead of the full container which might include various other modules (e.g. databases, servers etc). This is to make a lighter update and reduce exchanged data. The second step can also be a part of Continuous Integration cycle and be performed by using Jenkins for automatic deployment.

![Diagram of Docker containers](image)

**Figure 3: Integration and Update with Docker**

### 4.2.1 Docker Guidelines

A step by step guide has been created in order to support REVEAL developers while interacting with the Docker platform and runtime:

**Step 1:**

**Download a pre-built image**

# Download an ubuntu image

$ sudo docker pull Ubuntu:12.04

This will find the ubuntu image by name on Docker Hub [https://docs.docker.com/userguide/dockerrepos/](https://docs.docker.com/userguide/dockerrepos/) - find-public-images-on-docker-hub and download it from Docker Hub [https://hub.docker.com/](https://hub.docker.com/) to a local image cache.

**Step 2:**

**Running an interactive shell**

# Run an interactive shell in the ubuntu image,

# allocate a tty, attach stdin and stdout
# To detach the tty without exiting the shell,
# use the escape sequence Ctrl-p + Ctrl-q

$ sudo docker run -i -t -name <container_name> Ubuntu:12.04 /bin/bash

**Step 3:**

Listing containers

$ sudo docker ps # Lists only running containers

List running containers to verify it’s running.

**Step 4:**

Attach

Usage: docker attach [OPTIONS] CONTAINER

Attach to a running container

    --no-stdin=false  Do not attach STDIN

    --sig-proxy=true  Proxy all received signals to the process (even in non-TTY mode). SIGCHLD, SIGKILL, and SIGSTOP are not proxied.

Example:

$ sudo docker attach <container_id> OR <container_name>

The attach command will allow you to view or interact with any running container, detached (-d) or interactive (-i). You can attach to the same container at the same time - screen sharing style, or quickly view the progress of your daemonized process.

You can detach from the container again (and leave it running) with CTRL-q CTRL-p (for a quiet exit), or CTRL-c which will send a SIGKILL to the container, or CTRL-\ to get a stacktrace of the Docker client when it quits. When you detach from the container’s process the exit code will be returned to the client.

To stop a container, use docker stop.

To kill the container, use docker kill.

**Step 5:**

Committing (saving) a container state

Save your containers state to an image, so the state can be re-used.

When you commit your container only the differences between the image the container was created from and the current state of the container will be stored (as a diff). See which images you already have using the docker images command.

# Commit your container to a new named image
Step 6:

Account creation and login

Typically, you'll want to start by creating an account on Docker Hub (if you haven't already) and logging in. You can create your account directly on Docker Hub, or by running:

$ sudo docker login

This will prompt you for a user name, which will become the public namespace for your public repositories. If your user name is available, Docker will prompt you to enter a password and your e-mail address. It will then automatically log you in. You can now commit and push your own images up to your repos on Docker Hub.

Step 7:

Contributing to Docker Hub

Anyone can pull public images from the Docker Hub [https://hub.docker.com/](https://hub.docker.com/) registry, but if you would like to share your own images, then you must register first, as we saw in the first section.

4.3 Docker Private Registry

REVEAL private registry has been set up and running behind Nginx with HTTP basic authentication working. Since the connections were unencrypted the initial setup was not so secure. For this reason a self-signed SSL certificate was created in order to support SSL. Now we have a Docker registry running behind an Nginx server which is providing authentication and encryption via SSL.

A step by step guide that aims to support REVEAL developers while pushing custom Docker images to REVEAL private registry.

Pushing a repository to private REVEAL Docker Registry

In order to access the Reveal Docker private registry ([https://reveal.atc.gr:5001](https://reveal.atc.gr:5001)) please follow the instructions:

Create the certificate directory in your client machine:

$ mkdir /usr/local/share/ca-certificates/docker-dev-cert

Copy the SSL Certificate in the certificate directory.

Update the certificates:

$ update-ca-certificates

Restart Docker to make sure it reloads the system's CA certificates:
$ sudo service docker restart

Make sure that you have saved any changes in your containers before you restart them.

Now you should be able to log in to REVEAL private registry:

$ docker login https://reveal.atc.gr:5001

You may use the following credentials:

Username: reveal-dev
Password: ****

In order to come up with an image out of your Docker container:

commit any changes:

$ docker commit $[container_id] $[image-name]

You have to tag an image with the private registry's location in order to push to it:

$ docker tag $[image-name] reveal.atc.gr:5001/$[image-name]

push that image to our registry:

$ docker push reveal.atc.gr:5001/$[image-name]

### 4.4 Continuous Integration

**Description**

Continuous Integration is a software development practice where members of a team integrate their work frequently, leading to multiple integration cycles per day. On each cycle, the code is automatically tested to detect integration issues and the produced software modules are automatically deployed on an integrated testing environment. This approach leads to significantly reduced integration problems and allows a team to focus on adding features rather than spending time in bug fixing.

**Benefits**

- Eliminates long and tense integration phases – integration phases are relatively shorter
- Better collaboration and communication between teams of the project
- Fast detection and elimination of issues
- Less debugging, focusing on adding features
- Immediate testing of code in an integrated environment
- Deliver software more rapidly

---

7 [http://www.martinfowler.com/articles/continuousIntegration.html](http://www.martinfowler.com/articles/continuousIntegration.html)
Practices

- Maintain a single source repository (e.g. git/svn)
- Automate build and deployment with a CI server (e.g. Jenkins/Hudson)
- Make your build self-testing with unit tests (e.g. JUnit)
- Every commit should trigger a new build
- Deploy and test in a clone of the production environment
- Expose to the whole team the latest executable
- New build changes visible to everyone

Workflow

The workflow of the continuous integration process is summarized in the following diagram, starting from step 1 and repeating the same steps on every iteration.

![Diagram of REVEAL continuous integration workflow]

Figure 4: REVEAL continuous integration

Tools

Continuous Integration Workflow is described in detail in D6.1.1. The updated set of tools is described in the following section:
**Tools**

**Jenkins**: Jenkins is an open source tool for CI processes. It monitors the execution of repeated jobs, such as building a software project, focusing on building and testing software projects continuously and providing an easy-to-use so-called continuous integration system, making it easier for developers to integrate changes to the project, and making it easier for users to obtain a fresh build. The automated, continuous build increases the productivity. Jenkins can, also, monitor the execution of externally-run jobs, such as cron jobs and procmail jobs, even those that are run on a remote machine.

It is hosted at: [http://adrastea.atc.gr:8080/](http://adrastea.atc.gr:8080/) and is password protected.

Currently, there are 3 projects inside Jenkins:

![Figure 5: REVEAL projects in Jenkins](image)

**Apache Maven**: Apache Maven is a software project management and comprehension tool. Based on the concept of a Project Object Model (POM), Maven can manage a project's build, reporting and documentation from a central piece of information. It is important that all modules stay compatible with maven rules, sharing the same .pom file. In this way, the integration will be easily controlled through maven and moreover, the project will be independent of an IDE; as a maven project can be opened in Eclipse, Netbeans, IDEA etc.

**Redmine**: A flexible project management open source web application for issue tracking and ticketing.

The public REVEAL Redmine instance (v.2.1.2.stable.10669) is hosted at: [http://prometheus.atc.gr/projects/reveal/issues](http://prometheus.atc.gr/projects/reveal/issues)

It also supports the REVEAL wiki.

The Gantt diagram is showed in the following image:

---

8 http://jenkins-ci.org/

9 http://maven.apache.org/

10 http://www.redmine.org/
At the moment, the REVEAL Redmine community consists of 21 members:

**Gitlab**¹¹:

A software versioning and a revision control system distributed under a free license.

We have installed v.7.2.1 of Gitlab at a dedicated server: gitlab.atc.gr

We have also purchased and installed a signed certificate in order to secure the communication and avoid any malicious actions against it.

The gitlab web interface is exposed at: [https://gitlab.atc.gr/](https://gitlab.atc.gr/) and the REVEAL central repository is at: [https://gitlab.atc.gr/reveal/](https://gitlab.atc.gr/reveal/) and is password protected, accessible only by members of REVEAL community.

Currently, we have 11 projects inside gitlab:

---

¹¹ [https://subversion.apache.org/](https://subversion.apache.org/)
Currently, the REVEAL development community consists of 16 members:

<table>
<thead>
<tr>
<th>Members</th>
<th>Role</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stratos Tzoannos</td>
<td>Owner</td>
</tr>
<tr>
<td>Glorgos Glolis</td>
<td>Master</td>
</tr>
<tr>
<td>Symeon Papedapoulos</td>
<td>Developer</td>
</tr>
<tr>
<td>Georgios Katsimpras</td>
<td>Developer</td>
</tr>
<tr>
<td>Glorgos Rizos</td>
<td>Developer</td>
</tr>
<tr>
<td>Anastasia Krithara</td>
<td>Developer</td>
</tr>
<tr>
<td>Marc Dorchain</td>
<td>Developer</td>
</tr>
<tr>
<td>Vadims Krivoova</td>
<td>Developer</td>
</tr>
<tr>
<td>Wei Wang</td>
<td>Developer</td>
</tr>
<tr>
<td>Dimitrios Vogiatzis</td>
<td>Developer</td>
</tr>
<tr>
<td>Katerina Andreadou</td>
<td>Developer</td>
</tr>
<tr>
<td>Stuart Middleton</td>
<td>Developer</td>
</tr>
<tr>
<td>Stefanie Wiegand</td>
<td>Developer</td>
</tr>
<tr>
<td>Gerard Burnside</td>
<td>Developer</td>
</tr>
<tr>
<td>Stamatis Rapanakis</td>
<td>Developer</td>
</tr>
<tr>
<td>Paul Brigden</td>
<td>Developer</td>
</tr>
</tbody>
</table>

Figure 8: REVEAL projects in Gitlab

Figure 9: REVEAL Development Community
**SonarQube**\(^\text{12}\) :

SonarQube is an open source software quality platform. Sonar uses various static code analysis tools such as Checkstyle, FindBugs, Clover to extract software metrics, which then can be used to improve software quality.

Version 4.5.1 of SonarQube is installed at: [http://adrastea.atc.gr:9000/dashboard](http://adrastea.atc.gr:9000/dashboard)

**Sonatype Nexus**\(^\text{13}\) :

Nexus is an advanced repository management tool for controlling all the build and third party dependencies of a project. It can be used among a team of developers for sharing common libraries.

We have installed nexus sonatype at: [http://prometheus.atc.gr:8080/nexus/](http://prometheus.atc.gr:8080/nexus/)

For security purposes it is password protected and can be accessed only by REVEAL members. The reveal release repo is found at:

[http://prometheus.atc.gr:8080/nexus/content/repositories/releases/eu/reveal/](http://prometheus.atc.gr:8080/nexus/content/repositories/releases/eu/reveal/)

For the moment, only one project is deployed (jmongohandler):

![jmongohandler in Sonatype Nexus](image)

Figure 10: jmongohandler in Sonatype Nexus

In order to add a dependency located inside this repo inside the pom file of another project, the following repository has to be configured:

---


\(^{13}\) [http://www.sonatype.org/nexus/](http://www.sonatype.org/nexus/)
### 4.5 Implementation Process

In order to track all changes in the code, every new commit of code to the common repository has to be associated with a ticket in Redmine which is going to be characterized as “feature” or “bug”. Based on the milestones of the project, we are going to release stable versions in specific dates.

For both release management and ticket workflow, we have defined and agreed on the following policies also found in the REVEAL wiki:


### 4.6 The ticket workflow

1. Tickets assigned directly to feature owner (status=new) - very important to add type: feature/bug/support and a due date
2. If you don’t know the owner you can assign it to Stratos Tzoannos
3. The owner has to accept the ticket (status=in progress) or redirect it to other developer.
4. If developer fixes issue the ticket shall be redirected to the original submitter with status=resolved and description of the fix.
5. The original submitter is responsible to test and close the ticket.

### 4.7 How to define ticket priority

- Immediate (priority-1): system breaks
- Urgent (priority-2): feature not working
- High (priority-3): bug which doesn’t break core functionality of any feature
- Normal (priority-4): new feature or enhancement

### 4.8 Release Management

We have to declare a stable release prior to any major event (evaluation, review, milestone etc).

**10 days** before major event (evaluation, review) we **freeze development** - i.e. we stop committing in mainline. Any changes can be performed in code branches
Exceptions (commits are allowed):

- Changes for tickets with priority-1 (anytime)
- Changes for tickets with priority-2 (up to 5 days before event) with approval by project + technical coordinator

A release as stable if:

- System is up and running without any outage for at least 5 days.
- No priority-1 or priority-2 tickets exist in Redmine for this release
- No more than 5 priority-3 tickets exist in Redmine for this release

4.9 REVEAL Integration Roadmap

In this section we provide an overview of the integration plan in the REVEAL project based on the discussions followed with the technical partners and users.

Month 9 (Jul)

- Research in technologies/solutions
- Technical Discussions
- Specifications
- Definitions

Month 10 (Aug)

- Architecture V.1.0

Month 11 (Sep)

- Install Gitlab
- Create git accounts
- Setup CI environment (Hudson/maven/Sonar/Sonatype Nexus/Redmine (or Asana) + configuration (accounts, permissions)
- Setup Testbed with Python v.2.7, v3.3

Month 12 (Oct)

- Create project scaffolds in git repo (maven structure for each module)
- Install mongodb + configuration
- Install Solr + configuration (+ apache tomcat)
- Implement Solr API (HTTP or java+python libs)
- Implement mongodb API (HTTP or java+python libs)
- Implement crawler module for accessing contributor data for WP2 (from Twitter REST API)
- ITINNO deployment of ITINNO Crawler v.1.0 on the ATC testbed and testing by ITINNO
- Define orchestration mechanism for non-Storm workflows

Month 13 (Nov)

- HTTP API definition for endpoint controllers
- CI jobs + workflows (nightly builds etc)
- Implement crawler adaptor for getting items from RabbitMQ and store them to mongodb
- ITINNO/ATC run-through of the ITINNO crawler v.1.0 invoked by the ATC pilot app (or a mockup of it)
- ITINNO screen mockups (early ideas on paper) for map view, topic view, social view to allow end user partner comment
- ATC screen mockups (early ideas on paper) for UI

Month 14 (Dec)

- Implement orchestration mechanism for non-Storm workflows
- SAG Drupal service to fetch content from enterprise portal
- ATC gives guidelines to partners about how to install Docker at their local dev environment and (if needed) to provide Docker (Ubuntu) images to download and deploy

Month 15 (Jan)

- Preliminary prototype

Month 16 (Feb)

- Polling mechanism for fetching updates of content from SAG Drupal instance
- ATC test ITINNO 1st draft web page visualizations. Functionality will be basic at this stage and this is an integration test
- ATC test partner storm controllers invoked by ATC pilot app and the end user interface to do this.

Month 18 (Apr)

- Architecture v.2.0
- UniKob, ITINNO, CERTH to setup the Docker containers and provide them as images to ATC
- ITINNO screen mockups (based on in-development UI work) for map view, topic view, social view to allow end user partner comment
- ATC screen mockups (based on in-development UI work) for UI+filters

Month 19 (May)

- Deployment, configuration, testing of Docker images (in Storm clusters)
- ATC to deploy Docker images at REVEAL production cluster, run and test by ATC pilot app or mockup

Month 20 (Jun)

- ATC UI v.1.0 + filters
- Release of first prototype modules from each technical partner

Month 21 (Jul)

- Integration activities, debugging, testing of first prototype modules

Month 22 (Aug)

- Platform v.1.0
5 Presentation Layer

5.1 Interfaces

There will be two main interfaces in order to demonstrate all functionality from the REVEAL backend platform. The News Application and Enterprise Application web applications will comprise the core interface for REVEAL users. Moreover, the WP5 visualisation views will be hosted on a different environment but they will be integrated inside these web applications in order to ensure the maximum level of usability for the end users. Finally, the interface for the gamification features will be directly inserted into the Drupal platform of SAG Aris community to extend the existing portal with the related Drupal gamification modules.

5.2 Technologies/Dependencies

The two main interfaces will be implemented as web applications based on the following technologies:

5.2.1 AngularJS

AngularJS lets you extend HTML vocabulary for a specific application. The resulting environment is expressive, readable, and quick to develop.

It is a toolset for building the framework most suited to your application development. It is fully extensible and works well with other libraries. Every feature can be modified or replaced to suit unique development workflows and feature needs.

Data-binding in AngularJS is an automatic way of updating the view whenever the model changes, as well as updating the model whenever the view changes. This eliminates the need for DOM manipulation during the execution of a web application.

Controllers are the behavior behind the DOM elements. AngularJS allows to express the behavior in a clean readable form without the usual boilerplate of updating the DOM, registering callbacks or watching model change.

5.2.2 jQuery

jQuery is a free, open-source, cross-platform JavaScript library designed to simplify the client-side scripting of HTML. It is the most popular JavaScript library in use today.

jQuery's is designed to make it easier to navigate a document, select DOM elements, create animations, handle events, and develop Ajax applications. jQuery also provides capabilities to create plug-ins on top of the JavaScript library. This allows the creation of abstractions for low-level interaction and animation, advanced effects and high-level, theme-able widgets. The modular approach to the jQuery library allows the creation of powerful dynamic web pages and web applications.
5.2.3 Bootstrap

Bootstrap\(^{14}\) is currently the most popular HTML, CSS and JS framework for developing responsive, mobile projects on the web. By using bootstrap we offer the maximum user experience together with an impressive look-and-feel up of highest standard. The most important feature of bootstrap is that it allows the creation of a fully responsive user interface with an optimal viewing experience across a wide range of devices (mobile phones, tablets, PCs).

5.2.4 HTML5/CSS

HTML (the Hypertext Markup Language) and CSS (Cascading Style Sheets) are two of the core technologies for building Web pages. HTML provides the structure of the page, CSS the (visual and aural) layout, for a variety of devices. Along with graphics and scripting, HTML and CSS are the basis of building Web pages and Web Applications.

5.2.5 Decision support visualisation (ITINNO)

This is a module to be embedded in the UI (will be provided HTML + javascript code) to access and visualise the analysed output of the WP5. The description of this visualisation module is consolidated in the following table:

<table>
<thead>
<tr>
<th>Name</th>
<th>Decision support visualisation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>The WP5 visualisation component will be based upon a combination of a mapping server (e.g. Geoserver), database access layer and javascript. Each visualisation (i.e. geospatial, social and topical) is driven directly from the situation assessment database tables. In this way data is continually added / updated to the database and these changes are automatically visualized on the next refresh cycle. All visualisations are web-based and can be embedded into WP6 pilot applications.</td>
</tr>
<tr>
<td>Input</td>
<td>SQL queries of the situation assessment database table set (geospatial, social, topical) and SPARQL queries of trust model (triple store)</td>
</tr>
<tr>
<td>Output</td>
<td>Interactive HTML + Javascript (e.g. D3, OpenLayers) rendered web pages</td>
</tr>
<tr>
<td>API methods</td>
<td>N/A (driven by async database queries and visualising the latest situation assessment view)</td>
</tr>
<tr>
<td>Technologies</td>
<td>Python</td>
</tr>
<tr>
<td>3rd party dependencies</td>
<td>GeoServer, PostgreSql + PostGIS, HTML, Javascript - D3</td>
</tr>
</tbody>
</table>

The WP5 visualisations will be implemented based on D3 javascript library\(^{15}\) by performing the following workflow (assuming a standalone deployment of the visualisation web application):

1. Client opens browser and points to the needed visualisation HTML page i.e. http://192.9.206.91:8001/d3\_map\_view
2. During visualisation HTML loading all needed Javascript files (e.g. D3 etc.) will be loaded and the initial HTML page will be rendered (essentially HTML with several empty divs)
3. After the visualisation HTML will be fully loaded it will call needed Python visualisation backend (i.e. HTTP GET) in order to get data that will be needed to render/create the graphics i.e. data for

\(^{14}\) http://getbootstrap.com/

\(^{15}\) http://d3js.org/
map view or pie chart etc.
4. Python visualisation backend will query a database (query will be based on loaded visualisation) and it will parse the results (most commonly it will create a list of JSON objects) and will return data to the Viz HTML
5. When visualisation will receive all the data from the database it will render the graphics and create legend div
6. When user clicks on any of the ‘clickable’ parts on the visualisation - ‘Detailed View’ HTML div is automatically generated and appended to the document (on next click previous Detailed View div is simply overwritten with new one).

In order to integrate inside the News and Enterprise Web Applications, we are going to embed it inside the HTML of the News/Enterprise App page. So, when rendering the News/Enterprise App page, the visualisation service will be called (in an ajax call) in order to fetch the necessary elements for the page. The visualisation service will expose a RESTful GET method which returns a <div> element with the related HTML elements and javascript files inside. Then, the javascript will run to populate the internal elements with a mixture of SVG images and HTML elements. So, eventually, these elements will be inserted and will be part of the DOM of the main pages of the News/Enterprise Web Application UI. Since we need to be able to drag and drop the elements of the visualisations, we can simply ensure that dragable URL's (e.g. Twitter content items) are created by our visualisation code with HTML <div><a> elements. The drag and drop feature will be implemented by javascript.

The only problem that we have faced is the Cross Domain call blocking for the javascript of the visualisation, which is calling the visualisation backend in order to populate the various diagrams. Since it is called from inside the News/Enterprise Web App domain, it will not be allowed to call methods from a different domain. To overcome this problem, we are implementing a Cross-Origin Resource Sharing (CORS) mechanism by using a reverse proxy.

6 Business Layer

6.1 Crawling and Indexing

6.1.1 ITINNO crawler

This is the crawler for fetching social media content from the web to our platform to be used for analysis purposes. The Social Media Client Framework module (also referred as ITINNO crawler) has been installed on reveal.atc.gr server. Partners have provided it by pushing the corresponding Docker image that contains the module installation as well as all its dependencies like RabbitMQ server, postgreSQL etc, to Reveal private registry. Then, a Docker container was created by running the module image. The REST API provided by the Social Media Client Framework module has been integrated with the orchestrator mechanism so once a new assessment starts the ITINNO crawler gets notified and starts execution.

Access HTTP endpoint: http://reveal.atc.gr:49201

6.1.2 Focused Image Crawler (Web Crawler)
The focused image crawler is responsible for collecting images and their metadata from the Web given a set of keywords and a list of initial seed urls. A list of 800 international news sites is used at the moment, although the crawl soon expands to the whole Web by following outlinks from the landing pages. The basis of the crawler is the BUbiNG open source project (http://law.di.unimi.it/software/bubing-docs/overview-summary.html), which has been modified in order to explicitly, collect images related to the given set of keywords. To categorize an image as relevant to a subject or not, we check if at least some of the given keywords are contained in the image URL or in the alternate text of the HTML img element. Some additional checks are conducted once the image has been identified as relevant, for example minimum dimension constraints are imposed etc.

For every crawled image, all available metadata are parsed: the image width and height, the alternate text, the last modified date, the crawl date, the originating web page url and the text of the surrounding HTML elements. The data at hand can subsequently be stored in a database (mongo DB in our case), analyzed (for instance named entity extraction on the available text), clustered and indexed (in our case, SURF features are extracted from the image to construct a VLAD vector which is eventually indexed in a dedicated search structure).

Several aspects of the crawling process as well as the number of worker threads and hence the required memory can be configured by means of a configuration file loaded at startup but additionally a number of them are modifiable at runtime by using JMX.

6.1.3 WP2 crawler – Twitter RESTful API Crawler

A crawler has been developed that stores information retrieved from the Twitter API about a provided list of users. The main goal is to construct a hierarchy where user’s data are stored and information about other users who connect with a user (friends/ followers) are also retrieved and stored, up to a certain level. The application has been designed to handle the Twitter API limitations effectively and ensures the retrieval of all the users. The result is the availability of data that form a “community” of connected users, useful for further analysis.

**Operation**

The initial list consists of 50 accounts of Twitter active users, mainly journalists. They have been selected to confirm with actual use case scenarios, where each user of interest has a significant number of connections (friends/ followers). The input list can be configured to be read from an external database and should consist either of a Twitter username or a Twitter user id. For each user, the Twitter API is called to get information about his friends (accounts that the user follows) and followers (accounts who follow the user) and about himself (account creation date, number of tweets, description and others). After the computation of the input list, the list of users he follows is taken and for each of them the relevant data are retrieved and stored. This operation typically requires a lot of time since each user has thousands of friends and totally tens of thousands of users Twitter data will be retrieved. For practical reasons, a few dozens of users should be used as input and the configured level of computation (friends of friends) should not exceed 2. Furthermore, to make the computation faster, only the number of followers and not their actual ids can be configured to be retrieved for the last level of users.

**Twitter API rate limitations**
The connection with the Twitter API can be achieved either by authenticating as a user or as a registered Twitter application. Each mode has similar rate limits for the provided functions, as indicated in the Rate limits chart (https://dev.twitter.com/rest/public/rate-limits). We should notice the GET friends/list and GET followers/list functions which can be called only 15 times per 15-min time window. Each call returns at most 5000 IDs, which means that for a typical journalist with 72,000 followers, 15 calls to the GET followers/list function are required to get the full list of his followers. After that, the next call to the Twitter API (by using the same token) would raise a limitation exception, stating that no more requests are available. To overcome this problem, several user authentication tokens are used interchangeably. When ones remaining calls are depleted, the next available token is used. Since the list of available tokens is finite, when all the tokens have no remaining requests, the system sleeps for a time period. After that time period (typically 12 – 14 minutes), the execution is continued.

**Storage**

A mongo database is used to store the data and get the initial users list. Mongo is an open source NoSQL database that provides high performance and easy scalability. The fact that it is the most popular option among big data projects and its wide community support, made it the de facto choice for our application. The data are stored in a json format, therefore they can be easily manipulated by third party applications. The database connection strings are configured in a properties file, as well as the input and the result collections names. No recompiling of the code is needed. A script has been developed to ease the execution of the application in a Linux environment.

**Usage**

The data retrieved from the crawler can be reviewed by using a tool to connect to the Mongo database configured. Other applications that want to retrieve data can use a tool that has been developed for this purpose. The Mongo Handler API offers functions that permit the user to query the results database. For example the function getItemsWithQuery accepts as parameter a json query and returns the results in a json format. The tool supports main database operations and is available in Java and Python.

6.1.4 **SAG crawler (services to fetch data)**

The crawler of the SAG Enterprise Community is going to fetch data by calling the Web Services implemented in the scope of REVEAL for this purpose.

The specific web services for SAG Aris Drupal Enterprise Community are already implemented. The exposed RESTful resources are described as follows:

http://wp12123627.server-he.de/rest/reveal/reveal_endpoints_atc/retrieve?page=1&type=post

If .json is added after /retrieve a json file is returned instead. XML is default.

```
{"nid":"2952","cid":null,"reply_to":null,"comment_to":null,"indicator":0,"created":"1208075460","uid":"77","body_value":"
<p>Long-term business success […]</p>"},{"body_summary":"
<p>Long- […]efficiency.</p>"}
```

Nid: id of a post
cid: id of a comment or reply
reply_to: reference to reply where this reply belongs to
comment_to: reference to comment where this reply belongs to
indicator: 0 if it's a post, 1 if it's a comment 2 if it's a reply
created: unix timestamp
uid: user id  
boby_value: text of the body  
boby_summary: summary of the body  

[if !supportLists:] [endif]

http://wp12123627.server-he.de/rest/reveal/reveal_endpoints_atc/retrieve?page=1&type=userpost

includes all Fields of a post/comment/reply and the user who published this post/comment/reply.

This page delivers all profile information of a user:

http://wp12123627.server-he.de/rest/reveal/reveal_endpoints?page=1

- <uid>335130</uid>
- <roles>3,5,26,29</roles>
- Uid of the assigned roles
- <goals>2,1,4,5</goals>
- Gid of the assigned goals
- <name>mscheid</name>
- Unique username
- <created>1415621932</created>
  Unix timestamp
- <field_country_value>Germany</field_country_value
- <field_facebook_account_name_value>martin.scheid1@softwareag.com</field_facebook_account_name_value
- <field_linked_in_account_name_value>KeinLogin</field_linked_in_account_name_value
- <field_profession_value=test</field_profession_value
- <field_profile_about_value/>
- <field_profile_firstname_value>Martin</field_profile_firstname_value
- <field_profile_homepage_value>http://www.softwareag.com/</field_profile_homepage_value
- <field_profile_lastname_value>Scheid</field_profile_lastname_value
- <field_profile_organisation_value>SAG</field_profile_organisation_value
- <field_twitter_account_name_value>DAZZLE0</field_twitter_account_name_value
- <field_userpoints_value>1950</field_userpoints_value
- <field_xing_account_name_value>Martin_Scheid8</field_xing_account_name_value

Here are examples that include a timestamp:

Here are some examples for the syntax:

http://wp12123627.server-he.de/rest/reveal/reveal_endpoints_atc/retrieve?page=1&type=user&min_date=1242630202&max_date=1242630202

http://wp12123627.server-he.de/rest/reveal/reveal_endpoints_atc/retrieve.json?page=1&type=post&min_date=1242649800&max_date=1242649800

1242630202 is the timestamp as a unix timestamp.
&=type=post&min_date=1242649800&max_date=1242649800 retrieves the posts whose “created”
timestamp is: min_date <= timestamp <= max_date.

6.1.5 RabbitMQ2Mongo

The MQ2mongo adapter listens to RabbitMQ message queue and copies content to MongoDB, to be
subsequently used by batch processing jobs. The module has been installed on reveal.atc.gr server
as a system service. In order to make MQ2Mongo module to start on system startup an Upstart
script on reveal.atc.gr server has been created for this purpose. Thus, a convenient way to manage module’s initiation/termination follows:

    service mq2mongo start/stop/restart

### 6.2 Orchestration

#### 6.2.1 Offline Process Orchestrator (asynchronous)

For the modules that need to run in an asynchronous mode (long response times), we have implemented a mechanism to synchronise and monitor them. These modules mostly perform batch processing and are responsible for retrieving their input from mongodb (or RabbitMQ) and for pushing their output inside RabbitMQ (for further WP5 analysis and visualisation). If needed, they also store their output inside a different storage (e.g. mongodb). Their only interaction with the orchestrator is when being triggered by the orchestrator in order to start and when sending back an acknowledgment to the orchestrator on finishing.

**Synchronisation**

Since the modules of WP2 comprise of a number of Java and Python, non-Storm based, processes running in a number of servers there is the need to apply an orchestration mechanism in a distributed and efficient way. The proposed architecture consists of an orchestrator running anywhere in the REVEAL ecosystem and one controller running on each host server where the batch processing WP2 modules are running. Initially, the orchestrator gets triggered once a new assessment starts. Then, it encapsulates each module execution/task command as a message, appends the assessment id and sends it to the agreed exchange along with a routing key comprising of the server host name where the module has been installed and integrated. Note that the assessment id stands for the mongodb database name where the crawled items will end up. A worker process running in the background, on each controller, will pop the tasks and eventually execute the module execution job. Note the use of an exchange of type “direct”: this allows applying a routing algorithm based on the location of the modules. The orchestrator keeps a lookup table containing the mappings between the server host names and the command that triggers the execution of each module. The only requirement regarding the WP2 modules is to provide an executable (Java/Python) with a constructor to receive the assessment id, which reflects the newly generated MongoDB database/collection.

The modules may also accept as argument the exchange and routing_key of the orchestrator’s rabbitMQ as well as the timeslot start and end epoch times to define the timeslot of the batch to be processed. The batch for each module for each analysis cycle will be decided by the orchestrator and will be defined after experimenting on the execution of modules. For the moment it is decided based on an initial minimum batch needed by the modules to work (e.g. 10,000 tweets). After the first analysis finishes successfully for a module, the orchestrator will wait until a significant set of new tweets is collected and then it will reinitiate the module with either the updated set of data or the full content for the specific assessment (depending on whether the specific module supports an incremental analysis).

To tackle dependencies between modules and control the sequence of execution, in order to efficiently overcome any unnecessary complexity we will take advantage of a flagging mechanism in the repositories used in REVEAL architecture, especially in mongodb repository. The approach
consists of the enrichment of the original schema of the identified data objects with additional fields (flags) indicating a state in the chain of processing, such as "indexed", "analysed" and more. In that way the polling processes running in background will be able to identify and collect a precise subset of documents in the mongodb repository for further processing. An additional feature that will enhance the accuracy of the proposed flagging mechanism is that the mongodb repository allows any number of concurrent or parallel reads in the created collections.

Monitoring analysis modules

There is a need for the orchestrator to know the current status of the modules in order to synchronise them. For this reason, we are using a RabbitMQ message queue where each module sends status messages to a specific exchange. In this way, the orchestrator knows when a module has finished successfully or has failed analysis. Following a publish/subscribe pattern, the modules will be able to communicate with the orchestrator. The exchange name of the RabbitMQ broker will be passed as a parameter to the starting modules. This enables a fully distributed setup of the offline process.

The following diagram explains how the orchestrator synchronises two modules, which depend on each other (module-2 uses the output of module-1). As we can see, the main orchestrator invokes
first the module-1 through the orchestrator-client 1 and when the module-1 ends successfully, it sends back an asynchronous message to the orchestrator (via RabbitMQ pub/sub protocol). At second step the orchestrator invokes module-2. Then, once started, module-2 ensures that it only takes module-1 output by submitting the appropriate query to mongodb (e.g. use a flag inside the stored json object showing that module-1 output is present).

![Sequence Diagram](image)

Figure 13: Sequence Diagram for offline orchestration

### 6.2.2 Offline Process Orchestrator (synchronous)

For the modules of the analysis workflow that perform the processing rapidly, there is a need for a simpler way of synchronisation. Therefore, we have decided to implement these modules as java libraries which expose the necessary method to perform their analysis. In this way, the orchestrator is implemented as a java executable process (including as internal dependencies the related modules to be called) and calls one by one (or in parallel) the methods for performing the analysis pipeline. The input (e.g. set of tweets) is retrieved by the orchestrator and is passed as argument to the executed modules’ methods. Moreover, the output is returned to the orchestrator and is either forwarded to the next module of the pipeline or it is pushed into one of the storages (e.g. Apache Solr, mongodb, RabbitMQ).

### 6.2.3 Storm Orchestrator

**Synchronisation**

Synchronization is a rather critical issue in REVEAL since there are many dependencies in the chain of processing carried out either by the Storm modules. The RabbitMQ message broker could serve the goal of efficient synchronization between the REVEAL modules by issuing appropriate messages indicating the change in the state of the analysed/processed documents representing the various REVEAL data objects.
6.2.4 Communication

All outputs from WP2/3/4 will be published to a RabbitMQ broker. RabbitMQ will allow REVEAL partners to create a communication pipeline that enables loosely coupled processes to communicate asynchronously, scale with load, initiate system state changes, and report back on those changes as they occur. It’s a flexible system that can greatly enhance the speed and resiliency of complex system architectures, like the one designed for REVEAL. RabbitMQ is the leading open source implementation of the Advanced Message Queuing Protocol (AMQP). One of the things that REVEAL components can benefit from is that it can be built with a high level of fault tolerance.

In RabbitMQ messages are dropped on an exchange, and queues attach to that exchange to receive some subset of the messages. For the purposes of REVEAL components we will use a topic exchange, since in many cases one or many components will need to be provided with a set of already processed and analyzed messages. Topic exchanges route messages to one or many queues based on matching between a message routing key and the pattern that was used to bind a queue to an exchange. The topic exchange type is often used to implement various publish/subscribe pattern variations. Topic exchanges are commonly used for the multicast routing of messages. The queues lives in the RabbitMQ service bus, so listeners can be added and removed based on need.

In order to facilitate the communication between the various REVEAL components that will be emitting messages in JSON format and the RabbitMQ messages queues a client HTTP adapter to work with the REST interfaces will be created. The REST API provided by REVEAL REST Adapter will be used to produce and consume messages to and from RabbitMQ message queues.

6.3 Secondary Modules

There is a need for certain modules that do not take part in the analysis but participate in the workflow of WP6. Such modules can be considered as secondary since they provide support for invoking/controlling or scheduling main modules of the workflow.

- **MQ2mongo adapter**: this module will listen to RabbitMQ message queue and will copy content to mongodb, to be subsequently used by batch processing jobs.
- **Storm Controllers**: there will be one controller for each topology (ITINNO, UniKob, CERTH) to start a new storm topology when receiving a request through an HTTP interface.

More specifically, the controllers will provide the following API, as provided in deliverable D5.1:
HTTP endpoint: ITINNO controller

GET /itinno-controller/assessment/
GET /itinno-controller/assessment/<assessment-id>
  --> return status and active situation assessment URI's for visualization
POST /itinno-controller/assessment/<assessment-id>
  --> start search & crawling & replay
  --> stop
  --> add new focus area
  --> remove focus area
  --> add configuration
  --> remove configuration
DELETE /itinno-controller/assessment/<assessment-id>

HTTP endpoint: UniKob controller

GET /ukob-controller/assessment/
GET /ukob-controller/assessment/<assessment-id>
  --> return status and active situation assessment URI's for visualization
POST /ukob-controller/assessment/<assessment-id>
  --> start topic and semantic modelling
  --> stop
  --> add configuration
  --> remove configuration
DELETE /certh-controller/assessment/<assessment-id>

HTTP endpoint: CERTH controller

GET /certh-controller/assessment/
GET /certh-controller/assessment/<assessment-id>
  --> return status and active situation assessment URI's for visualization
POST /certh-controller/assessment/<assessment-id>
  --> start geolocation & image feature classification
  --> stop
  --> add configuration
  --> remove configuration
DELETE /certh-controller/assessment/<assessment-id>
6.4 Analysis Modules (module ID card)

6.4.1 Multimedia clusterer (CERTH)

<table>
<thead>
<tr>
<th>Name</th>
<th>Multimedia clusterer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Extracts clusters (groups) of images that are visually similar and hence depict the same scene / story</td>
</tr>
<tr>
<td>Input</td>
<td>Collection of images &amp; Metadata</td>
</tr>
<tr>
<td>Output</td>
<td>Image clusters with importance scores</td>
</tr>
<tr>
<td>API methods</td>
<td><code>/media/image/cluster</code> (POST)</td>
</tr>
<tr>
<td>Technologies</td>
<td>Java, a database (e.g. mongoDB) to store the cluster and metadata, BerkleyDB</td>
</tr>
<tr>
<td>3rd party dependencies</td>
<td>None</td>
</tr>
</tbody>
</table>

6.4.2 Multimedia geolocator (CERTH)

<table>
<thead>
<tr>
<th>Name</th>
<th>Multimedia geolocator</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Infers location information from multimedia items using a refined language model</td>
</tr>
<tr>
<td>Input</td>
<td>Image &amp; Metadata</td>
</tr>
<tr>
<td>Output</td>
<td>Location (latitude, longitude and higher level location information e.g. city / country) &amp; confidence score</td>
</tr>
<tr>
<td>API methods</td>
<td>LocationEstimate getLocationEstimate(Image img, Metadata metadata) where LocationEstimate will contain a Location and a double (confidence score)</td>
</tr>
<tr>
<td>Technologies</td>
<td>Java, a database to store the inferred location</td>
</tr>
<tr>
<td>3rd party dependencies</td>
<td>Storm</td>
</tr>
</tbody>
</table>

6.4.3 Multimedia indexing (CERTH)

<table>
<thead>
<tr>
<th>Name</th>
<th>Multimedia indexer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Indexes images by extracting local descriptors based solely on their visual content</td>
</tr>
<tr>
<td>Input</td>
<td>Image</td>
</tr>
<tr>
<td>Output</td>
<td>IndexingResult (probably a boolean saying whether indexing was successful or not)</td>
</tr>
<tr>
<td>API methods</td>
<td><code>/media/{collection_name}/index</code> (POST)</td>
</tr>
<tr>
<td>Technologies</td>
<td>Java, BerkeleyDB</td>
</tr>
<tr>
<td>3rd party dependencies</td>
<td>Storm</td>
</tr>
</tbody>
</table>

6.4.4 Multimedia similarity retriever (CERTH)

<table>
<thead>
<tr>
<th>Name</th>
<th>Multimedia similarity retriever</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Retrieves images (out of the set of indexed ones) that are very similar (near-duplicates) to the input image</td>
</tr>
<tr>
<td>Input</td>
<td>Image, collection to search</td>
</tr>
<tr>
<td>Output</td>
<td>Ranked list of near-duplicate images &amp; similarity scores</td>
</tr>
<tr>
<td>API methods</td>
<td><code>/media/image/search</code> (GET)</td>
</tr>
<tr>
<td>Technologies</td>
<td>Java, BerkeleyDB</td>
</tr>
<tr>
<td>3rd party dependencies</td>
<td>None</td>
</tr>
</tbody>
</table>
### 6.4.5 Multimedia Summariser (CERTH)

<table>
<thead>
<tr>
<th>Name</th>
<th>Multimedia summariser</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Picks posts that are characteristic in a large topic-focused set.</td>
</tr>
<tr>
<td>Input</td>
<td>Collection of social media posts.</td>
</tr>
<tr>
<td>Output</td>
<td>A set of N posts that are representative of the topic.</td>
</tr>
<tr>
<td>API methods</td>
<td>/media/image/summarize (POST)</td>
</tr>
<tr>
<td></td>
<td>/media/text/summarize (POST)</td>
</tr>
<tr>
<td>Technologies</td>
<td>Java 8, a database (e.g mongodb), graph database</td>
</tr>
<tr>
<td>3rd party dependencies</td>
<td>Lucene, stanford-nlp, s-space library</td>
</tr>
</tbody>
</table>

### 6.4.6 Multimedia manipulation detector (CERTH)

<table>
<thead>
<tr>
<th>Name</th>
<th>Multimedia manipulation detector</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Responsible for eliciting an estimation of the possible tampering operations an image has undergone</td>
</tr>
<tr>
<td>Input</td>
<td>A suspect image</td>
</tr>
<tr>
<td>Output</td>
<td>A number of confidence estimates, accompanied by localization maps, concerning the presence or absence of tampering traces.</td>
</tr>
<tr>
<td>API methods</td>
<td>ForensicAnalysis manipulationAnalysis (String imageUrl)</td>
</tr>
<tr>
<td>Technologies</td>
<td>Java, C, native open JPEG processing system libraries (JPEGLib),</td>
</tr>
<tr>
<td>3rd party dependencies</td>
<td>None</td>
</tr>
</tbody>
</table>

### 6.4.7 Fake Post Detector (CERTH)

<table>
<thead>
<tr>
<th>Name</th>
<th>Fake Post Detector</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Classifies a post with multimedia content (image, video) shared in social media, as fake (if shares untrustworthy and malicious information) or real (if shares trustworthy and verified information). It uses features from the text of the post and the post itself as well as features from the user shared the post.</td>
</tr>
<tr>
<td>Input</td>
<td>The id number of the post.</td>
</tr>
<tr>
<td>Output</td>
<td>A verification result for the post (fake or real and the fake probability as percentage).</td>
</tr>
<tr>
<td>API methods</td>
<td>VerificationResult tweetClassification (String id)</td>
</tr>
<tr>
<td>Technologies</td>
<td>Java, WEKA library</td>
</tr>
<tr>
<td>3rd party dependencies</td>
<td>None</td>
</tr>
<tr>
<td>Repositories</td>
<td>None</td>
</tr>
</tbody>
</table>

### 6.4.8 Multimedia social context builder and modality extractor (CERTH)

<table>
<thead>
<tr>
<th>Name</th>
<th>Multimedia Social Context Builder and Modality Extractor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Responsible for extracting social modalities for an online media item after creating a social interaction network around it.</td>
</tr>
<tr>
<td>Input</td>
<td>- Gets the Multimedia Social Context Builder module’s output for a given media item</td>
</tr>
</tbody>
</table>

---

**Project Title:** REVEAL  
**Contract No.:** FP7-610928  
**Project Coordinator:** INTRASOF International S.A.  
**www.revealproject.eu**
Output

- Provides numerical scores that quantify the popularity, influence and controversiality of media item
- Provides predicted numerical scores for popularity, influence and controversiality of media item

API methods

- collect_and_store(int media_item_id)
- dict modality_values
  quantify_modalities(int media_item_id)
- dict modality_predictions
  predict_modalities(int media_item_id)

The aforementioned dictionaries have strings as keys (e.g. “popularity”, “influence”, “controversiality”) and integers or floats as the corresponding values.

Technologies

- Python v3.4
- MongoDB
- OpenBLAS library

3rd party dependencies

OSN API wrapper for collecting social context

NOTE: The Multimedia social context builder has been merged with the Multimedia modality extractor and the new module is now being referred to as: Multimedia social context builder and modality extractor.

NOTE: The SN structure collector has been merged with the User network profile classifier

### 6.4.9 User network profile classifier (CERTH)

<table>
<thead>
<tr>
<th>Name</th>
<th>User Network Profile Classifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Responsible for deriving category information about a user based on the user’s network structure (explicit and implicit).</td>
</tr>
<tr>
<td>Input</td>
<td>Gets connection credentials to a Mongo database and instructions (e.g. time window and/or number of items) for reading raw social data documents in order to construct a user network. Alternatively, it must be given a set of category labels and some annotated users.</td>
</tr>
<tr>
<td>Output</td>
<td>Provides category predictions for the rest of the users with a prediction score</td>
</tr>
<tr>
<td>API methods</td>
<td>user_network_profile_classifier(mongo_db_credentials, twitter_api_credentials, rabbitmq_credentials)</td>
</tr>
<tr>
<td>Technologies</td>
<td>Python v3.4, RabbitMQ for sending success of failure messages, PServer, MongoDB, OpenBLAS library</td>
</tr>
<tr>
<td>3rd party dependencies</td>
<td>Twitter Search API for fetching twitter user lists.</td>
</tr>
</tbody>
</table>

### 6.4.10 Web crawler (CERTH)

<table>
<thead>
<tr>
<th>Name</th>
<th>Web crawler</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Crawls images and their metadata based on a set of keywords starting from a list of news sites as initial seeds</td>
</tr>
<tr>
<td>Input</td>
<td>A list of seed URLs &amp; the request keywords</td>
</tr>
<tr>
<td>Output</td>
<td>Indexed images and metadata</td>
</tr>
</tbody>
</table>
### 6.4.11 Community ranking (CERTH)

<table>
<thead>
<tr>
<th>Name</th>
<th>Community ranking</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>Discover track and rank dynamic communities</td>
</tr>
<tr>
<td><strong>Input</strong></td>
<td>A collection of Tweet like JSON formatted objects. Each entry should contain at least a connection between two users, a timestamp and textual content.</td>
</tr>
<tr>
<td><strong>Output</strong></td>
<td>JSON formatted objects containing a ranking of dynamic communities, a ranking of the most influential users within the communities, a category for each community and extensive textual and contextual information such as popular hashtags, keywords, bigrams and URLs for every dynamic and static community.</td>
</tr>
<tr>
<td><strong>API methods</strong></td>
<td>track_and_rank(dataset_Id,numOfTopCommunities,searchDepth)</td>
</tr>
<tr>
<td><strong>Technologies</strong></td>
<td>- Python v3.3</td>
</tr>
<tr>
<td></td>
<td>- NumPy, igraph, matplotlib, twython, pymongo, nltk, unshortenit, goslate.</td>
</tr>
<tr>
<td><strong>3rd party dependencies</strong></td>
<td>OSN API wrappers for collecting information</td>
</tr>
</tbody>
</table>

### 6.4.12 Geospatial context extraction (pre-) processing (ITINNO)

<table>
<thead>
<tr>
<th>Name</th>
<th>Geospatial context extraction (pre-) processing</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>Collection of Storm topologies supporting pre-processing and processing activities associated with geospatial context. The pre-processing topology takes new focus areas and prepares location information from a planet OpenStreetMap database ready for real-time geo-parsing. A geo-parse topology matches pre-processed locations to real-time text streams. A geo-classification topology classifies text streams in terms of how they use the location words.</td>
</tr>
<tr>
<td><strong>Input</strong></td>
<td>Prepared JSON from crawler via RabbitMQ + Planet OSM database + Geoclassifier training data</td>
</tr>
<tr>
<td><strong>Output</strong></td>
<td>JSON annotations to RabbitMQ (per content item geoparsed locations &amp; geoclassified class labels)</td>
</tr>
<tr>
<td><strong>API methods</strong></td>
<td>Storm topology (see D5.1 section 2)</td>
</tr>
<tr>
<td><strong>Technologies</strong></td>
<td>Python</td>
</tr>
<tr>
<td><strong>3rd party dependencies</strong></td>
<td>Storm, RabbitMQ, Openstreetmap, Postgresql + PostGIS</td>
</tr>
</tbody>
</table>

### 6.4.13 Situation assessment framework (ITINNO)

<table>
<thead>
<tr>
<th>Name</th>
<th>Situation assessment framework</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>The ‘situation assessment’ topology consists of a set of aggregation steps which combines all the incremental annotations into a single situation assessment database table set. New content items are added as new rows in an item table. New annotations add rows into tables for location, tags, URL's etc. When items or annotations are updated the associated statistics will also be recalculated. This situation assessment database table set can then be visualized by the decision support framework at any time.</td>
</tr>
<tr>
<td><strong>Input</strong></td>
<td>JSON annotations from WP2/3/4 components via RabbitMQ</td>
</tr>
<tr>
<td><strong>Output</strong></td>
<td>Situation assessment database table set (geospatial, social, topical) and SPARQL queries of trust model (triple store)</td>
</tr>
<tr>
<td><strong>API methods</strong></td>
<td>Storm topology (see D5.1 section 2)</td>
</tr>
</tbody>
</table>
6.1.2 – Architecture and specifications

Technologies

<table>
<thead>
<tr>
<th>Technologies</th>
<th>Python</th>
</tr>
</thead>
<tbody>
<tr>
<td>3rd party dependencies</td>
<td>Storm, Rabbitmq, OpenStreetMap, Postgresql + PostGIS, Triple store (e.g. OWLIM SE)</td>
</tr>
</tbody>
</table>

6.4.14 Social media client framework (ITINNO)

<table>
<thead>
<tr>
<th>Name</th>
<th>Social media client framework</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>HTTP endpoint supporting searching and crawling of various social media sites including Twitter, YouTube, Instagram and FourSquare. Crawled content is published to RabbitMQ as JSON formatted text ready for subsequent real-time and/or batch processing.</td>
</tr>
<tr>
<td>Input</td>
<td>public social media content (search/stream) and local content (replay)</td>
</tr>
<tr>
<td>Output</td>
<td>Fetched content</td>
</tr>
</tbody>
</table>
| API methods                       | GET /<media>  
|                                    | GET /<media>/<id>  
|                                    | POST /<media>/search  
|                                    | POST /<media>/stream  
|                                    | POST /<media>/replay  
|                                    | --&gt; replay a set of JSON objects serialized on disk  
|                                    | DELETE /<media>/stream/<id>  
|                                    | DELETE /<media>/replay/<id>  
|                                    | - tech ==&gt;  
|                                    | - 3rd party ==&gt; rabbitmq, postgresql + postGIS |
| Technologies                      | Python |
| 3rd party dependencies            | RabbitMQ, Postgresql + PostGIS |

6.4.15 Focused Geospatial exploration (UniKob)

<table>
<thead>
<tr>
<th>Name</th>
<th>Focused Geospatial exploration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>The module extends the spatial semantic context of a semantically encoded and interlinked entity, e.g. a user or a social media content item. This means, for an entity represented on the Linked Open Data cloud as a URI the module explores the links of this entity and identifies additional and related location entities. These additional location entities enrich the geospatial context.</td>
</tr>
<tr>
<td>Input</td>
<td>URI of a location entity</td>
</tr>
<tr>
<td>Output</td>
<td>URIs of related location entities</td>
</tr>
<tr>
<td>API methods</td>
<td></td>
</tr>
<tr>
<td>Technologies</td>
<td>Java, RDF</td>
</tr>
</tbody>
</table>
| 3rd party dependencies            | Geospatial Context extraction Module (ITInno), which encodes recognized Open Street Map entities as LinkedGeoData URIs.  
|                                    | External Java libraries for working with RDF (currently JENA) |
| Repositories                      | RDF-Triplestore (e.g. Sesame, Virtuoso) |

6.4.16 Geospatial topic model (UniKob)

<table>
<thead>
<tr>
<th>Name</th>
<th>Geospatial topic model</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>The module builds a region-aware topic model. It is a supervised probabilistic approach which needs training data to detect regions and the topics. Afterwards it can be applied to identify a probability distribution of</td>
</tr>
</tbody>
</table>
6.1.2 Architecture and specifications

| topics and regions over a given content item. Accordingly the module will provide two different API interfaces: one for training the model and one for analysing content items. |

| Input | Training phase:  
| | • Set of content items (text) annotated with geospatial coordinates  
| | Application phase:  
| | • Content item (text)  |

| Output | Training phase:  
| | • Regions  
| | • Topic model (Probability distributions of words over regions)  
| | Application phase:  
| | • Topic distribution  
| | • Region distribution  |

| API methods | N/A |
| Technologies | Java |
| 3rd party dependencies | None (currently, maybe some convenience Java libraries) |
| Repositories | None (currently, maybe a repository will be used to serialize the topic models) |

6.4.17 Role analysis (UniKob)

| Name | Role analysis |
| Description | The module will assign roles to users based upon their interaction activity pattern in a specific community. The module is unsupervised, but needs a reference data set to automatically detect activity levels. Therefore it has API interfaces for an unsupervised training phase as well as for the subsequent application of generated classification rules. |

| Input | Training phase  
| | • Set of feature vectors for users. The vector has 7 dimensions and float values.  
| | Application phase  
| | • Single user feature vector (same style as for training)  |

| Output | Training phase  
| | • Boundaries of activity levels for high, med or low activity for each feature dimension  
| | Application phase  
| | • One of the following 8 roles: Elitist, Grunt, Joining Conversationalist, Popular Initiator, Popular Participant, Supporter, Taciturn, Ignored  |

| API methods | N/A |
| Technologies | Java |
| 3rd party dependencies | • PServer (NCSR’D): for storing roles  
| | • Social Context Module (UKob): feature generation  
| | • External Java libraries for working with statistics (Apache commons (math))  |
| Repositories | None (currently) |

6.4.18 Semantic context model (UniKob)

| Name | Semantic context model |
| Description | The module will construct and analyse the semantic context of semantic entities. To this end it will use publicly available information from the Linked Open Data cloud to connect the entities on a semantic level and evaluate the shape and type of this connections, e.g. by assigning a coherence score. Accordingly the module will have two different API outputs, one for the simple construction of the connecting semantic graph and one for the analysis. |
### 6.4.19 Social context model (UniKob)

#### Name
Social context model

#### Description
The module analyses the interaction between users and content items to construct social context features for a specific community. These features quantify levels of interaction as well as aggregated characteristics of a user and/or content items. These features serve as basis for other analytical processes. The module will be flexible to operate on different types of communities. In its most basic form it only assumes the availability of structural information consisting of users and discussion trees without actually any information about the content. If a concrete analysed community provides more information (e.g. Twitter, StackExchange, etc.), the module will be extended to cover also this information.

#### Input
- Community consisting of users and content items as well as an author connection between users and content items and a reply relation between content items.
- (Optional) additional features of users and content items, e.g. user age or the actual text in a content item.

#### Output
- Feature vectors for users
- Feature vectors for content items
- Feature vectors for discussion trees

#### API methods
N/A

#### Technologies
Java

#### 3rd party dependencies
- External Java libraries for working with time, statistics, etc. (json.org, Apache commons (math), Apache commons (cli), Joda-time)

#### Repositories
None (currently, maybe a repository will be used to retrieve the raw community information)

### 6.4.20 Topical context model (UniKob)

#### Name
Topical context model

#### Description
This module is a generalization of the geospatial topic module. It will derive topic modules for text based on several influential factors from the context (e.g. authors, communities, time). As the geospatial topic module, it will comprise a training phase and an application phase.

#### Input
- Training phase:
  - Set of content items (text) with additional context information
- Application phase:
  - Content item (text)

#### Output
- Training phase:
  - Topic model (Probability distributions)
- Application phase:
  - Topic distribution for an item

#### API methods
N/A

#### Technologies
Java
6.4.21 Source finding (Albf)

<table>
<thead>
<tr>
<th>Name</th>
<th>Source finding</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>This module would exploit the same method (in an inverse mode) than the one used to measure visibility (extended influence) for trying to find the original provenance of a tweet, if it was a bit modified and does not appear as a classical retweet.</td>
</tr>
<tr>
<td>Input</td>
<td>Tweets</td>
</tr>
<tr>
<td>Output</td>
<td>A specific contributor’s tweet</td>
</tr>
<tr>
<td>API methods</td>
<td>N/A</td>
</tr>
<tr>
<td>Technologies</td>
<td>• Java</td>
</tr>
<tr>
<td></td>
<td>• Json as the tweet format, although we can adapt to another format</td>
</tr>
<tr>
<td>3rd party dependencies</td>
<td>• External Java libraries for working with JSON (currently Google library GSON)</td>
</tr>
<tr>
<td>Repositories</td>
<td>• We will align with crawlers outputs to define whether we need specific repositories or databases. We are not constrained by the storage so far.</td>
</tr>
</tbody>
</table>

6.4.22 Influence calculation (Albf)

<table>
<thead>
<tr>
<th>Name</th>
<th>Influence calculation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>The module will compute several different values, say popularity, global influence, topic-sensitive influence or extended influence. To summarize, <strong>popularity</strong> would describe the direct neighbourhood (one hop away) in the social network graph. A more <strong>global influence</strong>, i.e. beyond the first hop, can be computed by taking into account retweets and replies. <strong>Topic-sensitive influence</strong> expresses the level of influence of a contributor regarding a specific topic, it may take into account the number of tweets and interactions about a specific topic. An <strong>extended version of influence</strong> could also measure a kind of <strong>visibility</strong>, i.e. how the information contained in the user tweets and conversations is propagated discussed and modified beyond the first circle that is one hop away.</td>
</tr>
<tr>
<td>Input</td>
<td>Tweets</td>
</tr>
<tr>
<td>Output</td>
<td>Different values depending of the function, to add information about the contributor (whose profile should be stored in the Pserver)</td>
</tr>
<tr>
<td>API methods</td>
<td>N/A</td>
</tr>
<tr>
<td>Technologies</td>
<td>• Java</td>
</tr>
<tr>
<td></td>
<td>• Json as the tweet format, although we can adapt to another format</td>
</tr>
<tr>
<td>3rd party dependencies</td>
<td>• External Java libraries for working with JSON (currently Google library GSON)</td>
</tr>
<tr>
<td>Repositories</td>
<td>• We will align with crawlers outputs to define whether we need specific repositories or databases. We are not constrained by the storage so far.</td>
</tr>
</tbody>
</table>

6.4.23 Contributor alignment (Albf)

<table>
<thead>
<tr>
<th>Name</th>
<th>Contributor alignment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>We speak about contributor alignment to describe how the topics that are discussed by the contributor or influencer are a lot discussed elsewhere in</td>
</tr>
</tbody>
</table>
the social network beyond the sphere of influence of the user, i.e. if they are hot topics with respect to some user-defined criteria. The level of alignment depends on the list of hot topics that the contributor also speaks about.

<table>
<thead>
<tr>
<th>Input</th>
<th>Tweets</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output</td>
<td>The level of alignment of the contributor (whose profile should be stored in the Pserver)</td>
</tr>
<tr>
<td>API methods</td>
<td>N/A</td>
</tr>
<tr>
<td>Technologies</td>
<td>• Java</td>
</tr>
<tr>
<td></td>
<td>• Json as the tweet format, although we can adapt to another format</td>
</tr>
<tr>
<td>3rd party dependencies</td>
<td>• External Java libraries for working with JSON (currently Google library GSON)</td>
</tr>
<tr>
<td>Repositories</td>
<td>• We will align with crawlers’ outputs to define whether we need specific repositories or databases. We are not constrained by the storage so far.</td>
</tr>
</tbody>
</table>

### 6.4.24 Multipartite Community detection (NCSR)

<table>
<thead>
<tr>
<th>Name</th>
<th>Multipartite Community discovery</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Discovering communities of users, tags and URLs. These are 3 entities that form a three partite network. Each multipartite community will contain nodes from all entities. Other possibilities exist, for instance we may have users, named entities, and relations between named entities. The end result is information rich communities</td>
</tr>
<tr>
<td>Input</td>
<td>• Processed social network data that comprise users, tags and resources (e.g. a URL or an image), other possibilities exist: for instance users that posts, users that reply and keywords used or even users, named entities and relations between named entities</td>
</tr>
<tr>
<td></td>
<td>• Input might be fed also from the modules of WP3 that produce named entities and relations</td>
</tr>
<tr>
<td>Output</td>
<td>• Communities of users, tags and resources or communities of uses, named entities and relations</td>
</tr>
<tr>
<td></td>
<td>• The above information will be made available in JSON format</td>
</tr>
<tr>
<td>API methods</td>
<td>N/A</td>
</tr>
<tr>
<td>Technologies</td>
<td>Has been rewritten in Java</td>
</tr>
<tr>
<td>3rd party dependencies</td>
<td>Stanford java libraries for Named Entity Recognition</td>
</tr>
<tr>
<td>Repositories</td>
<td>MongoDB will be needed to store community profiles</td>
</tr>
</tbody>
</table>

### 6.4.25 Influence calculation (NCSR)

<table>
<thead>
<tr>
<th>Name</th>
<th>Influence Calculation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Topic-Specific Supervised Random Walks (TS-SRW) is a supervised learning algorithm for identifying topic-specific influential users. Thus a network of user interactions (e.g. mentions), the content they have produced, and a given list of influential users the algorithm will learn to</td>
</tr>
</tbody>
</table>
discover new topic specific influential users.

**Input**
At the moment, the algorithm needs as input raw social network data (which will be pre-processed to obtain the implied mention graph and the textual content)

**Output**
A ranked list of users (user_id: influence_score)

**API methods**
N/A

**Technologies**
Python 2.7

**3rd party dependencies**
At the moment:
- Python libraries: scipy, numpy, networkx, nltk

**Repositories**
Results will be stored in PServer (MySQL)

### 6.4.26 Pserver (NCSR)

**Name**
PServer Profiles

**Description**
Representation and maintenance of user (and possibly) community profiles in PServer. (Modalities and communities are computed by modules that communicate with PServer).

**Input**
- Raw Social network data (e.g. number of followers)
- Input from WP2 software modules (e.g. user influence or user role)

**Output**
- User/Contributor profile: comprising demographic data, modalities (e.g. influence), user roles (e.g. elitist), etc.
- Communities and community profiles
- The above information will be made available in JSON format (each user profile will be sent as JSON file)

**API methods**
N/A

**Technologies**
Java, MySQL

**3rd party dependencies**
None

**Repositories**
It needs a MySQL database to operate

### 6.4.27 Relation discovery (NCSR)

**Name**
Relation Discovery

**Description**
The module will identify relations between entities within a text (e.g. "Obama went to China": "Obama" and "China" are the entities and "went" is the relations between them).

**Input**
JSON as input
<table>
<thead>
<tr>
<th>Output</th>
<th>JSON as output</th>
</tr>
</thead>
<tbody>
<tr>
<td>API methods</td>
<td>N/A</td>
</tr>
<tr>
<td>Technologies</td>
<td>Java (possibly Python)</td>
</tr>
<tr>
<td>3ʳᵈ party dependencies</td>
<td>For now, no Storm will be used (maybe during the second year, if it’s necessary)</td>
</tr>
<tr>
<td>Repositories</td>
<td>Solr, MongoDB</td>
</tr>
</tbody>
</table>

### 6.4.28 Semantic segmentation (NCSR)

<table>
<thead>
<tr>
<th>Name</th>
<th>Semantic segmentation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>The module cleans an html from all tags and returns the cleaned text of an article</td>
</tr>
<tr>
<td>Input</td>
<td>html page as input</td>
</tr>
<tr>
<td>Output</td>
<td>cleaned text as output</td>
</tr>
<tr>
<td>API methods</td>
<td>N/A</td>
</tr>
<tr>
<td>Technologies</td>
<td>Java</td>
</tr>
<tr>
<td>3ʳᵈ party dependencies</td>
<td>For now, no Storm will be used (maybe during the second year, if it’s necessary)</td>
</tr>
<tr>
<td>Repositories</td>
<td>MongoDB</td>
</tr>
</tbody>
</table>

### 6.4.29 Stylometry (NCSR)

<table>
<thead>
<tr>
<th>Name</th>
<th>Stylometry</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>The module will identify the author profile</td>
</tr>
<tr>
<td>Input</td>
<td>JSON as input</td>
</tr>
<tr>
<td>Output</td>
<td>JSON as output</td>
</tr>
<tr>
<td>API methods</td>
<td>N/A</td>
</tr>
<tr>
<td>Technologies</td>
<td>Java (possibly Python)</td>
</tr>
<tr>
<td>3ʳᵈ party dependencies</td>
<td>For now, no Storm will be used (maybe during the second year, if it’s necessary)</td>
</tr>
<tr>
<td>Repositories</td>
<td>MongoDB</td>
</tr>
</tbody>
</table>

### 6.4.30 Presence (NCSR)
### Name
- Presence (WP2>T2.3)

### Description
- Responsible for identifying duplicate accounts of a user on different social media.

### Input
- Contributor Profiles from social network 1
- Community Profiles from social network 2

### Output
- An array of tuples, each containing two user account ids, in different social networks, followed by a probability of the duplicate identity of two accounts

### API methods
- N/A

### Technologies
- Java

### 3rd party dependencies
- Opensource libraries in Java for string matching

### Repositories
- Results will be stored in PServer (MySQL)

---

**6.4.31 Community Tracking (NCSR)**

<table>
<thead>
<tr>
<th>Name</th>
<th>Community Tracking</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>Tracking communities of users across time. (e.g. user community 1 grew from the 1\textsuperscript{st} day to the 2\textsuperscript{nd} or user community 2 became extinct in the same period)</td>
</tr>
<tr>
<td><strong>Input</strong></td>
<td>Social network data that comprise a network of user interactions</td>
</tr>
<tr>
<td></td>
<td>Segmentation of data in time frame</td>
</tr>
<tr>
<td><strong>Output</strong></td>
<td>Communities per time frame, and correspondence between communities of neighbouring time frames (e.g. grows(c1,c2,t1\rightarrow t2), it denotes that c1 grew and become c2 from t1 to 2; splits(c6,c7,c8,t1\rightarrow t2) denotes that c6 split into c7 and c8)</td>
</tr>
<tr>
<td><strong>API methods</strong></td>
<td>N/A</td>
</tr>
<tr>
<td><strong>Technologies</strong></td>
<td>It has been rewritten in Python</td>
</tr>
<tr>
<td><strong>3rd party dependencies</strong></td>
<td>Depends on the NetworkX library of Python</td>
</tr>
<tr>
<td><strong>Repositories</strong></td>
<td>MongoDB to store results</td>
</tr>
</tbody>
</table>

---

**6.4.32 Complex Event Recognition (NCSR)**

Project Title: REVEAL
Project Coordinator: INTRASOF International S.A.
<table>
<thead>
<tr>
<th>Name</th>
<th>Complex event recognition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Recognise higher levels modalities with the aid of logic based</td>
</tr>
<tr>
<td></td>
<td>rules that are crisp or probabilistic. For example,</td>
</tr>
<tr>
<td></td>
<td>trust(user, topic, high) if</td>
</tr>
<tr>
<td></td>
<td>influence (user, topic, high) and</td>
</tr>
<tr>
<td></td>
<td>influence holds True for a certain time period</td>
</tr>
<tr>
<td></td>
<td>In the future it could be extended for other modalities</td>
</tr>
<tr>
<td>Input</td>
<td>Raw social network data, and input from lower level</td>
</tr>
<tr>
<td></td>
<td>modalities, e.g. influence of user, and possibly other</td>
</tr>
<tr>
<td></td>
<td>modalities such as the user's role.</td>
</tr>
<tr>
<td></td>
<td>It depends on an annotated data set.</td>
</tr>
<tr>
<td>Output</td>
<td>An estimated of a higher level modality for each user (e.g.</td>
</tr>
<tr>
<td></td>
<td>an estimate of trustworthiness)</td>
</tr>
<tr>
<td>API methods</td>
<td>N/A</td>
</tr>
<tr>
<td>Technologies</td>
<td>Currently the rules run in RTEC (an inference engine for</td>
</tr>
<tr>
<td></td>
<td>event calculus) and in LOMRF (an inference engine for</td>
</tr>
<tr>
<td></td>
<td>probabilistic rules that are based on Markov Logic Networks).</td>
</tr>
<tr>
<td>3rd party dependencies</td>
<td>RTEC, LOMRF: both are open source projects</td>
</tr>
<tr>
<td>Repositories</td>
<td>PServer to store the higher level modality value of a user</td>
</tr>
</tbody>
</table>

### 6.4.33 Community Evolution Prediction (NCSR)

<table>
<thead>
<tr>
<th>Name</th>
<th>Community Evolution Prediction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>User communities in dynamic networks evolve over time, they</td>
</tr>
<tr>
<td></td>
<td>may grow, shrink or disappear. This module aims to predict</td>
</tr>
<tr>
<td></td>
<td>what a user community will do in the near future (this is in</td>
</tr>
<tr>
<td></td>
<td>contrast to the community tracking module that tries to</td>
</tr>
<tr>
<td></td>
<td>examine the past of a community as some sort of forensics)</td>
</tr>
<tr>
<td></td>
<td>There is a training and an operational phase.</td>
</tr>
<tr>
<td></td>
<td>Training phase: given some input (i.e. communities) and the</td>
</tr>
<tr>
<td></td>
<td>correct output (i.e. evolutionary label, e.g. grow, shrink)</td>
</tr>
<tr>
<td></td>
<td>the module is trained to predict the evolution of</td>
</tr>
<tr>
<td></td>
<td>communities. Operation phase: once the module is trained</td>
</tr>
<tr>
<td></td>
<td>it will try to prediction the future of the currently</td>
</tr>
<tr>
<td></td>
<td>discovered communities.</td>
</tr>
<tr>
<td>Input</td>
<td>A set of tweets represented as json files in MongoDB, a</td>
</tr>
<tr>
<td></td>
<td>definition of time frame (start and end point)</td>
</tr>
<tr>
<td>Output</td>
<td>In the operational phase it will attach an evolutionary</td>
</tr>
<tr>
<td></td>
<td>phenomenon to every community discovered. E.g. (c1, grow),</td>
</tr>
<tr>
<td></td>
<td>(c2, disappear) meaning that</td>
</tr>
</tbody>
</table>
6.4.34 Linguistic analysis (NSCR)

<table>
<thead>
<tr>
<th>Name</th>
<th>Linguistic analysis</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>The module will identify the entities within a text (e.g. &quot;Obama went to China&quot;: &quot;Obama&quot; and &quot;China&quot; are the).</td>
</tr>
<tr>
<td>Input</td>
<td>JSON as input</td>
</tr>
<tr>
<td>Output</td>
<td>JSON as output</td>
</tr>
<tr>
<td>API methods</td>
<td>N/A</td>
</tr>
<tr>
<td>Technologies</td>
<td>Java (possibly Python)</td>
</tr>
<tr>
<td>3rd party dependencies</td>
<td>For now, no Storm will be used (maybe during the second year, if it's necessary)</td>
</tr>
<tr>
<td>Repositories</td>
<td>MongoDB, Solr</td>
</tr>
</tbody>
</table>

6.4.35 Trust and credibility model (Ukob-ALCATEL)

<table>
<thead>
<tr>
<th>Name</th>
<th>Trust and credibility</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Trust (or distrust) might be the result of social interactions between users. Certain interactions might generate trust, others might destroy trust. This module will build a supervised model for trust prediction based on interaction features for tuples of users. This means, it will look at a pair of user and their interaction patterns, compare these observations to other interaction patterns of users for which it is known if they do or do not trust each other and attempt to predict the trust for this user pair. The approach is supervised, which motivates two APIs, one for training, one for application.</td>
</tr>
</tbody>
</table>
| Input                 | Training phase:  
|                       | • Set of feature vectors for pairs of users and the label whether or not they trust each other.  
|                       | Application phase:  
|                       | • Feature vector for a user pair  
| Output                | Training phase:  
|                       | • Trust model  
|                       | Application phase:  
|                       | • Prediction of trust, either binary (yes/no) or as a score (float)  
| API methods           | Java  
| Technologies          | Java |
### 3\textsuperscript{rd} party dependencies

- Social Context Module (UKob): feature generation
- maybe some suitable Java libraries (Weka, Apache Commons, etc.)

### Repositories

None (currently, maybe a local repository will be used to serialize the trust models)

---

#### 6.4.36 Trust and credibility model (ITINNO)

<table>
<thead>
<tr>
<th>Name</th>
<th>Trust and credibility (ITINNO)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Evidential knowledge-based approach to trust and credibility modelling. Real-time incremental evidence can be asserted as OWL facts in a triple store (by the situation assessment component). In addition end user analysts can asset situation specific a-priori knowledge. Scalable simple inferences will classify large volumes of evidence and support interactive analysis of situation assessment views from individual analysts' perspectives. This work includes a UI to support interactive analysis sessions with the trust and credibility model.</td>
</tr>
<tr>
<td>Input</td>
<td>Evidence from situation assessment + a-priori situation specific knowledge</td>
</tr>
<tr>
<td>Output</td>
<td>Inferred classifications of evidence allowing filtering/ranking/analysis of situation assessments by analysts</td>
</tr>
<tr>
<td>API methods</td>
<td>OWL inference patterns + UI for interactive analysis</td>
</tr>
<tr>
<td>Technologies</td>
<td>Triple store (e.g. OWLIM SE) + HTML + Javascript (e.g. D3) + Python backend data layer</td>
</tr>
<tr>
<td>3\textsuperscript{rd} party dependencies</td>
<td>ITINNO situation assessment + ITINNO Decision support visualisation</td>
</tr>
<tr>
<td>Repositories</td>
<td>Triple store (e.g. OWLIM SE)</td>
</tr>
</tbody>
</table>

---

#### 6.4.37 Sentiment Analysis (ATC)

<table>
<thead>
<tr>
<th>Name</th>
<th>Sentiment Analysis Classifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Identifies if a text has positive, negative or neutral sentiment. Designed especially for sentiment analysis of short texts (tweets).</td>
</tr>
<tr>
<td>Input</td>
<td>Gets the text whose sentiment has to be identified.</td>
</tr>
<tr>
<td>Output</td>
<td>Provides category predictions of the sentiment of the text (positive, negative, neutral)</td>
</tr>
<tr>
<td>API methods</td>
<td>Sentiment findSentiment(String text)</td>
</tr>
<tr>
<td>Technologies</td>
<td>Java</td>
</tr>
<tr>
<td>3\textsuperscript{rd} party dependencies</td>
<td>None</td>
</tr>
</tbody>
</table>
7  Data Access - Persistence Layer

7.1  Data access objects

7.2  APIs – python/Java for MongoDB

We have created 2 wrapper APIs for mongodb – one in Java and another one in Python.

7.2.1  Java wrapper API for MongoDB

The Java Wrapper code is found inside gittlab:  [https://gitlab.atc.gr/reveal/jmongohandler.git](https://gitlab.atc.gr/reveal/jmongohandler.git)

It is build inside Jenkins and is deployed at Nexus Sonatype. It can be added as maven dependency with the following pom section:

```xml
<dependency>
  <groupId>eu.reveal</groupId>
  <artifactId>jmongohandler</artifactId>
  <version>1.0</version>
  <classifier>dependencies</classifier>
</dependency>
```

The online javadoc of this API can be found here:


The API description is described with the following method signatures:

<table>
<thead>
<tr>
<th>Modifier and Type</th>
<th>Method and Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>static MongoHandler</td>
<td>valueOf(String type)</td>
</tr>
<tr>
<td>static MongoHandler[]</td>
<td>values()</td>
</tr>
</tbody>
</table>
/**
 * Using a cursor to get all the documents.
 *
 * @param dbName
 * @param collectionName
 * @return
 */

public DBCursor getItems(String dbName, String collectionName)

/**
 * Getting a set of documents with a query.
 *
 * @param dbName
 * @param collectionName
 * @param query
 * @return
 */

public DBCursor getItemsWithQuery(String dbName, String collectionName,
                   DBOBJECT query)

/**
 * Inserting a document.
 *
 * @param dbName
 * @param collectionName
 * @param documentToInsert
 * @return
 */

public WriteResult insert(String dbName, String collectionName,
                   DBOBJECT documentToInsert)

/**
 * Executes all the operations in parallel and aggregates up all the errors.
 * Unordered bulk operations do not guarantee order of execution.
 */
public BulkWriteResult bulkInsertUnordered(String dbName, String collectionName, List<DBObject> documentsToInsert)

/**
 * Executes all the operation in order and error out on the first write error.
 * 
 * @param dbName
 * @param collectionName
 * @param documentsToInsert
 * @return
 */

public BulkWriteResult bulkInsertOrdered(String dbName, String collectionName, List<DBObject> documentsToInsert)

/**
 * Remove documents from a collection.
 * 
 * @param dbName
 * @param collectionName
 * @param deletionCriteria: the deletion criteria using query operators.
 * Omit the query parameter or pass an empty document to delete all documents in the collection.
 * @return
 */

public WriteResult remove(String dbName, String collectionName,
**DBObject deletionCriteria**

`/**
* Modifies an existing document or documents in collection. By default the
* method updates a single document.
*
* @param dbName
* @param collectionName
* @param updateConditionsDocument: the selection criteria for the update
* @param updateOperationsDocument: the modifications to apply
* @param upsertFlag: when true, inserts a document if no document matches
* the update query criteria
* @param multiFlag: when true, updates all documents in the collection that
* match the update query criteria, otherwise only updates one
* @return
*/`

```java
public WriteResult update(String dbName, String collectionName,
                          DBOBJECT updateConditionsDocument, DBOBJECT updateOperationsDocument,
                          boolean upsertFlag, boolean multiFlag)
```

### 7.2.2 Python wrapper API for MongoDB

A python API to access mongodb has been implemented and is located inside gitlab repository: [https://gitlab.atc.gr/reveal/pymongohandler.git](https://gitlab.atc.gr/reveal/pymongohandler.git)

It is also deployed at a public Python registry as a package:

[https://pypi.python.org/pypi/pymongohandler/0.1](https://pypi.python.org/pypi/pymongohandler/0.1)

It is important to see that we have already 44 downloads in the first month (not only by REVEAL community members).
Figure 14: REVEAL python mongo handler description

The API description is described as follows:

```python
# returns a Cursor instance, which allows to iterate over all matching documents.

def getItems(self):
    # can pass a query document to limit/filter the returned documents
    # returns a Cursor instance, which allows to iterate over all matching documents.

def getItemsWithQuery(self, query):

    # Inserts a document or documents into a collection.

    def insert(self, document):

    # Unordered bulk write operations are batched and sent to the server in arbitrary order where they may be executed in parallel.
    # Any errors that occur are reported after all operations are attempted.

    def bulkInsertUnordered(self, documentsList):

    # Ordered bulk write operations are batched and sent to the server in the order provided for serial execution.
    # The return value is a document describing the type and count of operations performed.

    def bulkInsertOrdered(self, documentsList):

    # an update conditions document to match the documents to update,
    # an update operations document to specify the modification to perform, and
    # an options document.

```
7.3 Storm RabbitMQ Data Access (Java/Python rabbitMQ client)

The data access of RabbitMQ message broker will be handled by the available client libraries of RabbitMQ in Java and Python:

http://www.rabbitmq.com/devtools.html#java-dev
http://www.rabbitmq.com/devtools.html#python-dev

7.4 Apache Solr Data Access

The Apache Solr Data Access will be performed through the RESTful API provided by Solr. There are plenty of free libraries that encapsulate this API in various languages. For instance: http://wiki.apache.org/solr/SolrJ for Java or http://wiki.apache.org/solr/SolPython for Python. Once we have some finalised and concrete collections and specified methods for searching and indexing we can implement a Java concrete library for this purpose.

7.5 Common Data Transfer Objects (wiki)

In REVEAL wiki we have started an attempt to maintain the structure of all exchanged data transfer objects: http://prometheus.atc.gr/projects/reveal/wiki/common

For the moment we are covering the following three structures:

- Basic post object (Item) as retrieved by the ITINNO crawler
- Basic post object (Item) as exposed by SAG (Aris) community
- User Info object as exposed by SAG (Aris) community

The detailed json schemas are found in the Appendix of this document.
7.6 Repositories

7.6.1 MongoDB
MongoDB is an open-source document database that provides high performance, high availability, and automatic scaling.

In MongoDB, a record is a document. A document is a data structure composed of field and value pairs. MongoDB documents are similar to JSON objects. The values of fields may include other documents, arrays, and arrays of documents.

Using documents as records provides significant advantages. First of all documents (i.e. objects) correspond to native data types in many programming languages. Secondly embedded documents and arrays often reduce I/O activity on the database system and the need for expensive joins. Fluent polymorphism is also supported thanks to the dynamic schema offered by MongoDB.

To guarantee high availability, MongoDB’s replication facility, called replica sets, provide automatic failover and data redundancy. A replica set is defined as a group of MongoDB servers that maintain the same data set, providing redundancy and increasing data availability.

From a scalability point of view MongoDB provides horizontal scalability as part of its core functionality. Automatic sharding distributes data across a cluster of machines and replica sets can provide eventually-consistent reads for low-latency high throughput deployments.

7.6.2 Apache Solr
Solr\textsuperscript{16} is an open source enterprise search server based on the Lucene Java search library. Some of the key features it offers are: advanced full-text search, optimized for high volume web traffic content, scalability through efficient replication, faceting information retrieval and more. Due to its scalable nature we believe it fits best inside the high scalable architecture of REVEAL ecosystem. Additionally, Solr offers near real-time search (near real-time search means that documents are available for search almost immediately after being indexed) so it can handle the time critical retrieval of the analyzed and context annotated information provided by WP4/5 modules.

\textsuperscript{16} Apache Solr: \url{http://lucene.apache.org/solr/}
8 Physical Deployment

8.1 Hardware Specifications

Farm of IBM x3550 M4 servers with the following specs:

- CPU: 2 * E5-2620v2
- RAM: 192 GB
- FC connectivity to an array of IBM v5000 Midrange storage, with IBM “Easy Tier” technology enabled.

ATC has implemented a cloud infrastructure combining with virtualization for all its production services as well as development needs. Our software stack for providing cloud infrastructure includes, but not limited to, OpenStack, Hyper-V 2012 R2 and VMware products.

In the infrastructure described above, ATC has integrated an open source portal for Cloud Computing to automate the provisioning of a full suite of services on Windows servers, that also supports provisioning of Windows and Linux Virtual Machines.

We use a combination of Orchestrators based on the cloud provision suitable to the infrastructure be provided with commercial and open-source software.

ATC’ private cloud, consists of powerful Intel platform servers, suitable high-performance storage devices, firewall hardware and software and networking equipment combining technologies to provide performance and flexibility.

ATC’s cloud solutions offers flexible, dynamic and efficient configuration to provide the best solution according to specific computing needs.

The Virtualized infrastructure implemented was designed to provide high availability for critical services, as well as scalability for performance intensive applications.

Therefore it is based on:

1. H/W that supports scaling up:

   - Memory resources (a quarter of the supported memory is currently installed)
   - Network connectivity
   - Storage expandability (one sixth of the chassis that can be supported is currently installed)

2. The hypervisor technologies chosen can easily scale out, nearly fifteen times the current installation in the same cluster, for each cluster.

Supporting both scalability options have the following benefits:

   - Scale-out works better for CPU-intensive tasks since there are more cores
   - Scale-up works better for shuffle intensive tasks since it has fast intermediate storage and no network bottleneck

Depending on the future needs, we can choose either to scale up or scale out our infrastructure.
8.2 Infrastructure

We are going to deploy the various modules in the following topology of servers:

- 4 servers (2 CPU, 32GB Ram, 800GB HD) for ITINNO Storm topology as follows:
  - server 1
    - OSM/our_databases + nimbus/zookeeper/topologies + controller + visualisation
  - server 2
    - OSM/our_databases + supervisor
  - server 3
    - OSM/our_databases + supervisor
  - server 4
    - OSM/our_databases + supervisor

- 1 server (2 CPU, 32GB Ram, 800GB HD) for social media client framework (i.e. crawler) + rabbitmq broker
- 2 servers (2 CPU, 32GB Ram, 800GB HD) for WP3 Storm topology
- 2 servers (2 CPU, 32GB Ram, 800GB HD) for WP4 Storm topology
- 1 server (2 CPU, 32GB Ram, 800GB HD) for the Web Crawler (Focused Image Crawler)
- 2 servers (2 CPU, 16GB Ram, 400GB HD) for the WP2 analysis workflow (and offline asynchronous orchestrator)
- 1 server (2 CPU, 16GB Ram, 400GB HD) for the Geo Server and UI
- 1 server (2 CPU, 16GB Ram, 400GB HD) for Apache Solr + WP3 synchronous analysis process (+ offline synchronous orchestrator)
- 1 server (2 CPU, 16GB Ram, 400GB HD) for mongodb
9 Testing

The REVEAL platform will be tested in the context of WP6 to assess the maturity of the technical implementation and the alignment to the user requirements from a technical perspective. The technical assessment of the REVEAL platform is supported by monitoring the technical parameters of the platform performance and aims to determine how far the integrated prototype meets the technical requirements and the functional specifications.

REVEAL development tasks are tested according to established standards on software assurance process. This process aims to assess the efficiency of the platform functionalities and provide evidence that the integrated prototype is fully functional and available for release through a software assurance process. In principle, software assurance can be realised by evaluating both the software itself (the product) and how it has been developed (the process). Both aspects are important for a platform targeting to support scalable functionalities.

However, in prototypes such as the REVEAL platform, the software assurance of the process becomes quite difficult. This is mostly due to the constant changes in both the design and the requirements of the software during the project cycle itself, as a result of the ongoing work in the other work packages and the living lab approach used for the user requirements gathering and the user engagement process. Thus, the software assurance process means testing the outcome of the prototype is preferred, considering this as an integrated platform consisting of individual components which are integrated using the architecture principles that are analysed in this document.

Software validation is the “confirmation by examination and provision of objective evidence that software specifications conform to user needs and intended uses, and that the particular requirements implemented through software can be consistently fulfilled”. Since software is usually part of a larger hardware system, the validation of software typically includes evidence that all software requirements have been implemented correctly and completely.

In general, software validation is the process of developing a “level of confidence” that the system meets all requirements, functionalities, and user expectations as set out during the design process. It is a critical tool used to assure the quality of its component and the overall system. It allows for improving/refining the end product.

Software validation is realised through quality models. In the past, different quality models have been proposed, each of which addresses different quality attributes that allow evaluating the developed software. Some of the most well-known are:

- McCall's model of software quality\(^\text{17}\) (GE Model, 1977), which incorporates 11 criteria encompassing product operation, product revision, and product transition.
- Boehm's spiral model\(^\text{18}\) (1978) based on a wider range of characteristics, which incorporates 19 criteria. The criteria in both, this and the GE model, are not independent as they interact with each other and often cause conflicts.


\(^{18}\) Barry W. Boehm, “ A spiral model of software development and enhancement”, TRW, Defence Systems Group
• ISO 9126-1\(^{19}\) incorporates six quality goals, each goal having a large number of attributes. These six goals are then further split into sub-characteristics, which represent measurable attributes (custom defined for each software product).

Recently the BS ISO/IEC 25010:2011 standard\(^{20}\) about system and software quality models has replaced ISO 9126-1. Applying any of the above models is not a straightforward process. There are no automated means for testing software against each of the characteristics defined by each model. For each model, the final attributes must be matched against measurable metrics and thresholds for evaluating the results must be set. It is then possible to measure the results of the tests performed (either quantitative or qualitative/observed).

For the REVEAL case, we have adopted the ISO/IEC 25010:2011 standard, which is the most widespread reference model and it includes the common software quality characteristics that are supported by the other models. This standard defines two quality models providing a consistent terminology for specifying, measuring and evaluating system and software product quality:

- Quality in use model, which is composed of five characteristics that relate to the outcome of interaction with the system and characterises the impact that the product can have on the stakeholders.
- Product quality model, which is composed of eight characteristics that relate to static properties of software and dynamic properties of the computer system.

For our case, the product quality model is adopted. The eight characteristics, are further divided into sub-characteristics, as shown in the following figure:

Figure 15: The ISO/IEC 25010:2011 system/software quality model characteristics

For each of the sub-characteristics, a metric/measurable attribute is defined, along with thresholds. These metrics and thresholds are customised for each software product, which in our case is the REVEAL platform (consisting of individual components). By evaluating the complete set of metrics, we will be able to assess the overall quality of our platform and the percent to which we were able to meet the user requirements (reflected to system specifications and functionalities) defined during the design phase of the project.

---


\(^{20}\) BS ISO/IEC 25010:2011, Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - System and software quality models
10 Conclusion

The current document contains the architecture specifications and design of the integrated REVEAL platform that will serve as a basis for the development tasks of the project. This architecture description document will be very useful to define and communicate the initial architecture of the REVEAL platform. The architecture will continue to evolve throughout the project and we need to make sure that it is consistent with design and implementation work in the other technical work packages and the early pilot activities of the project.

This deliverable presented the activities for the delivery of a baseline implementation of the REVEAL platform, which acts as the reference point for the actual development of this platform and offers a shared and common background for the Consortium participants on the envisaged technologies that are necessary to build such a platform.
# APPENDIX A: REVEAL vocabulary

The list below presents the REVEAL vocabulary at its current state. This will be enhanced throughout the project lifetime.

<table>
<thead>
<tr>
<th><strong>Collection</strong>:</th>
<th>Content that can be browsed and searchable through a search API as a logical unit.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Community</strong>:</td>
<td>A contributor cluster with additional information such as the relations between its members.</td>
</tr>
<tr>
<td><strong>Community Profile</strong>:</td>
<td>Meta-information related to a community, such as relevant tags, prominent users, community size etc.</td>
</tr>
<tr>
<td><strong>Confidence Score</strong>:</td>
<td>A numerical value quantifying the belief of a statement or an algorithm result.</td>
</tr>
<tr>
<td><strong>Connected Contributor</strong>:</td>
<td>The user linked with a relationship within a Social Network (friend, follower etc.).</td>
</tr>
<tr>
<td><strong>Contributor</strong>:</td>
<td>The user of a social network (Facebook, twitter etc.).</td>
</tr>
<tr>
<td><strong>Contributor Category</strong>:</td>
<td>refers to the Social Network User Class (?) as written in the components description. The word “Class” is reserved in most OO programming languages.</td>
</tr>
<tr>
<td><strong>Contributor Cluster</strong>:</td>
<td>A group of users with similar characteristics.</td>
</tr>
<tr>
<td><strong>Contributor Influence Score</strong>:</td>
<td>Numeric value that indicates the calculated influence of a Contributor.</td>
</tr>
<tr>
<td><strong>Contributor Profile</strong>:</td>
<td>The profile information of a Contributor’s account.</td>
</tr>
<tr>
<td><strong>Contributor Topic Influence (Score?)</strong>:</td>
<td>Numeric value that refers to a combination of user and topic. It indicates the influence of a user upon a specific topic.</td>
</tr>
<tr>
<td><strong>Contributor Trustworthiness (Score?)</strong>:</td>
<td>Numeric value that indicates the reliability score of a Social Network User.</td>
</tr>
<tr>
<td><strong>Crawling Request</strong>:</td>
<td>A request sent to a crawling module to invoke a new crawl.</td>
</tr>
<tr>
<td><strong>Crawling</strong>:</td>
<td>The act of accessing Web servers, APIs or file systems in order to extract information to feed into the search platform. By following links, a crawler is able to traverse Web content hierarchies based on a single-start URL.</td>
</tr>
<tr>
<td><strong>Crowdsourcing</strong>:</td>
<td>Is the practice of obtaining needed services, ideas, or content by soliciting contributions from a large group of people, and especially from an online community.</td>
</tr>
<tr>
<td><strong>Gamification</strong>:</td>
<td>Is the use of game thinking and game mechanics in non-game contexts to engage users in solving problems.</td>
</tr>
<tr>
<td><strong>GeoLocation</strong>:</td>
<td>The geographic location of an entity (user/event/topic) expressed ideally by its coordinates and its name or by a higher level location description (e.g. city name)</td>
</tr>
<tr>
<td><strong>Item</strong>:</td>
<td>A piece of information that describes a retrievable logical unit. The indexed items within the search index constitute a Collection. In REVEAL an Item can be considered a piece of text in a Social Network e.g. “StatusUpdate” or a “tweet” associated with its metadata (author, date etc.). Similarly, in the case of OfficialSources the Item will be the textual content of a story/article, linked (conceptually) with its metadata.</td>
</tr>
<tr>
<td><strong>Item Metadata</strong>:</td>
<td>The additional information that describes an Item (e.g. StatusUpdate, GeoLocation, Landmark, Translations, Author, Date etc.). Programmatically, the metadata will reside inside the Item Class.</td>
</tr>
<tr>
<td><strong>Item Metadata Field</strong>:</td>
<td>Each field of the item metadata. It can be of varying type (text, string, datet ime etc.).</td>
</tr>
<tr>
<td><strong>Item Popularity Score</strong>:</td>
<td>Numeric value that indicates the popularity of an item (e.g. calculation of retweets, likes, rating etc.). The popularity could also be expressed with a more complex structure (e.g. a collection of ratings from different social media platforms) instead of a numeric score.</td>
</tr>
<tr>
<td><strong>Item Cluster</strong>:</td>
<td>A group of items clustered according to certain requirements (typically as a result of a clustering process)</td>
</tr>
<tr>
<td><strong>Media Item</strong>:</td>
<td>Any multimedia item (in the context of REVEAL use cases, multimedia items refer to images and videos).</td>
</tr>
<tr>
<td><strong>Media Item Authenticity Score</strong>:</td>
<td>A numeric value that indicates the authenticity of a</td>
</tr>
</tbody>
</table>
multimedia item. It is used for characterising fake images/videos.

**Media Item Category:** refers to the Media Item Class as written in the components description. The word “Class” is reserved in most OO programming languages. The category refers to the nature of the Media Item (e.g. photograph, diagram, rendering).

**Media Item Concept:** refers to the context, scene or object being depicted by the Media Item (e.g. nature, sea, car). An interesting sub-category is the Geographic Concept (e.g. building, desert, weather phenomena).

**Media Manipulation:** In the context of REVEAL use cases, media manipulation is the post-processing of a media item (image or video) after it has been originally captured, in order to create an illusion, a forgery or to express a specific opinion by adding or removing information from the media content.

**Mining:** Finding useful facts in databases of text or multimedia content; evaluating large amounts of stored data and looking for useful patterns.

**Named Entity:** a phrase that clearly identifies one item from a set of other items that have similar attributes. Examples of named entities are first and last names, geographic locations, ages, addresses, phone numbers, companies and addresses. Named entities are often mined for marketing initiatives.

**Official Source:** An informative Website to be crawled for a given domain. For example, in the case of news (BBC, the Guardian, DW, die Zeit), in the case of cinema (IMDb, rotten tomatoes).

**Processing Pipeline:** Sequential stages of processing within the search engine before the creation of a final index of the content.

**REVEAL User:** The user of the REVEAL platform.

**REVEAL User Profile:** The profile information of the REVEAL user.

**Search Result:** A result (URL and title) returned after a query is submitted to a Search Engine. It can also possibly be a more complex structure (including score etc.).

**Search Result Set:** A list of search results (Search Result items).

**Social Network Object:** Twitter user, Facebook page, YouTube channel.

**Stylometry:** Technique for capturing important features of the style of contributors

**Topic:** A story or event of interest e.g. Facebook event, twitter hash tag.

<table>
<thead>
<tr>
<th>Table 1: REVEAL vocabulary</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Media Item Category:</strong> refers to the Media Item Class as written in the components description. The word “Class” is reserved in most OO programming languages. The category refers to the nature of the Media Item (e.g. photograph, diagram, rendering).</td>
</tr>
<tr>
<td><strong>Media Item Concept:</strong> refers to the context, scene or object being depicted by the Media Item (e.g. nature, sea, car). An interesting sub-category is the Geographic Concept (e.g. building, desert, weather phenomena).</td>
</tr>
<tr>
<td><strong>Media Manipulation:</strong> In the context of REVEAL use cases, media manipulation is the post-processing of a media item (image or video) after it has been originally captured, in order to create an illusion, a forgery or to express a specific opinion by adding or removing information from the media content.</td>
</tr>
<tr>
<td><strong>Mining:</strong> Finding useful facts in databases of text or multimedia content; evaluating large amounts of stored data and looking for useful patterns.</td>
</tr>
<tr>
<td><strong>Named Entity:</strong> a phrase that clearly identifies one item from a set of other items that have similar attributes. Examples of named entities are first and last names, geographic locations, ages, addresses, phone numbers, companies and addresses. Named entities are often mined for marketing initiatives.</td>
</tr>
<tr>
<td><strong>Official Source:</strong> An informative Website to be crawled for a given domain. For example, in the case of news (BBC, the Guardian, DW, die Zeit), in the case of cinema (IMDb, rotten tomatoes).</td>
</tr>
<tr>
<td><strong>Processing Pipeline:</strong> Sequential stages of processing within the search engine before the creation of a final index of the content.</td>
</tr>
<tr>
<td><strong>REVEAL User:</strong> The user of the REVEAL platform.</td>
</tr>
<tr>
<td><strong>REVEAL User Profile:</strong> The profile information of the REVEAL user.</td>
</tr>
<tr>
<td><strong>Search Result:</strong> A result (URL and title) returned after a query is submitted to a Search Engine. It can also possibly be a more complex structure (including score etc.).</td>
</tr>
<tr>
<td><strong>Search Result Set:</strong> A list of search results (Search Result items).</td>
</tr>
<tr>
<td><strong>Social Network Object:</strong> Twitter user, Facebook page, YouTube channel.</td>
</tr>
<tr>
<td><strong>Stylometry:</strong> Technique for capturing important features of the style of contributors</td>
</tr>
<tr>
<td><strong>Topic:</strong> A story or event of interest e.g. Facebook event, twitter hash tag.</td>
</tr>
</tbody>
</table>
12 APPENDIX B: Basic post object (Item) as retrieved by the ITINNO crawler.

```json
{   "contributors": null,   "truncated": false,   "text": "@Say3d_Sal3m shit that get more deeply in iraq we are about to bring the war to our street muslims kill muslims Arabians kill Arabians",   "in_reply_to_status_id": 5.3502893309626e+17,   "favorite_count": NumberInt(0),   "source": "<a href="http:\/\ируется\download\iphone" rel="nofollow">Twitter for iPhone</a>",   "retweeted": false,   "coordinates": null,   "timestamp_ms": "1416396106272",   "entities": {      "user_mentions": {         "0": {            "id": NumberInt(303272905),            "indices": {               "0": NumberInt(12)            },            "id_str": "303272905",            "screen_name": "Say3d_Sal3m",            "name": "Sayed Salem"         },         "trends": [            ],         "hashtags": [            ],         "urls": [            ],         "in_reply_to_screen_name": "Say3d_Sal3m",         "id_str": "535030164577460225",         "retweet_count": NumberInt(0),         "in_reply_to_user_id": NumberInt(303272905),         "favorited": false,         "user": {            "follow_request_sent": null,            "profile_use_background_image": true,            "default_profile_image": false,            "id": NumberInt(324972506),            "verified": false,            "profile_image_url_https": "https:\/\%\/pbs.twimg.com\profile_images\/528188799750205444\Y91gWzTMNormal.jpeg",            "profile_sidebar_fill_color": "252429",            "profile_text_color": "666666",            "followers_count": NumberInt(1648),            "profile_sidebar_border_color": "000000",            "id_str": "324972506",            "profile_background_color": "1A1B1F",            "listed_count": NumberInt(2),            "profile_background_image_url_https": "https:\/\%\/pbs.twimg.com\profile_background_images\/378800000058078761\c2e8f2c20976a602887c3fd7e6dbb30.jpeg",            "utc_offset": NumberInt(-25200),            "statuses_count": NumberInt(5723),            "description": ".............",            "friends_count": NumberInt(664),            "location": "Mokkatm._.",            "profile_link_color": "3B94D9",            "profile_image_url": "https:\/\%\/pbs.twimg.com\profile_images\/528188799750205444\Y91gWzTMNormal.jpeg",            "following": null,            "geo_enabled": true,            "profile_banner_url": "https:\/\%\/pbs.twimg.com\profile_banners\/324972506\1377293192",            "profile_background_image_url": "https:\/\%\/pbs.twimg.com\profile_background_images\/378800000058078761\c2e8f2c20976a602887c3fd7e6dbb30.jpeg",            "name": "DemonicZoldier",            "lang": "en",            "profile_background_tile": false,            "favourites_count": NumberInt(358),            "screen_name": "Offline_e",            "notifications": null,            "url": null,            "created_at": "Mon Jun 27 14:39:12 +0000 2011",            "contributors_enabled": false,            "time_zone": "Arizona",            "protected": false,            "default_profile": false,            "is_translator": false         },         "geo": null,         "in_reply_to_user_id_str": "303272905",         "possibly_sensitive": false,         "lang": "en",         "created_at": "Wed Nov 19 11:21:46 +0000 2014",         "filter_level": "medium",         "in_reply_to_status_id": "535028933096236381",         "place": null,         "_id": ObjectId("546c7dd5a05970a6870024e6")    }```
13 Appendix C: Basic post object (Item) as exposed by SAG Aris community services

```json
{
    "nid": "9022",
    "cid": null,
    "comment_to": null,
    "reply_to": null,
    "indicator": "0",
    "Post_created": "1415264915",
    "Puid": "319343",
    "body_value": "How can I change the font size in ARIS Business Publisher. The view is not the same as in Business Designer/Architect?",
    "body_summary": "",
    "Uuid": "319343",
    "roles": null,
    "goals": null,
    "name": "sdizdar",
    "Member_since": "1403267995",
    "field_country_value": "Turkey",
    "mail": "devnull_319343@ariscommunity.com",
    "field_facebook_account_name_value": null,
    "field_linkedin_account_name_value": null,
    "field_profile_firstname_value": "Sema",
    "field_profile_lastname_value": "Dizdar",
    "field_profile_organisation_value": "Anadolu Bank A.",
    "field_twitter_account_name_value": null,
    "field_userpoints_value": null,
    "field_xing_account_name_value": null
}
```
14 APPENDIX D: User info object as exposed by SAG Aris community services

```json
{
  "users_name": "KOY",
  "uid": "335131",
  "users_created": "1415621932",
  "users_mail": "Konstantin.Yershov@softwareag.com",
  "users_language": "",
  "users_login": "0",
  "users_signature": "",
  "About you": [],
  "ARIS Express Registered": "",
  "ARIS MashZone registered": "",
  "Country": [],
  "Facebook Accountname": [],
  "First Name": [],
  "Homepage": [],
  "Last Name": [],
  "linkedin Accountname": [],
  "Organisation": [],
  "Receive ARIS Product Updates": "",
  "Receive Community Updates": "",
  "Receive ARIS Express Updates": "",
  "Title": "Ms.",
  "Twitter account name": [],
  "University Relations Group": "",
  "Xing Accountname": []
}
```