**REVEAL**

**FP7-610928**


**REVEALing hidden concepts in Social Media**

# Deliverable D5.1

# Analytic report of metadata template definitions

| | |
|---|---|
| **Editor(s):** | Stuart E. Middleton |
| **Responsible Partner:** | University of Southampton IT Innovation Centre |
| **Status-Version:** | Final - 1.1 |
| **Date:** | 29/08/2014 |
| **EC Distribution:** | Public |

| | |
|---|---|
| **Project Number:** | FP7-610928 |
| **Project Title:** | REVEAL |

| | |
|---|---|
| **Title of Deliverable:** | Analytic report of metadata template definitions |
| **Date of Delivery to the EC:** | 29/08/2014 |

| | |
|---|---|
| **Workpackage responsible for the Deliverable:** | WP5 - Modalities Analysis Framework |
| **Editor(s):** | Stuart E. Middleton (ITINNO) |
| **Contributor(s):** | Paul Walland, Vadims Krivcovs (ITINNO)<br>Thomas Gottron (UKob)<br>Katerina Andreadou, Georgios Rizos, Symeon Papadopoulos (CERTH)<br>Dimitrios Vogiatzis (NCSR'D) |
| **Reviewer(s):** | UKob |
| **Approved by:** | All Partners |

| | |
|---|---|
| **Abstract:** | This deliverable outlines the approach and structure used within WP5 for receiving 'self-described' modality reports from WP2/3/4. A JSON metadata format has been chosen and this deliverable provides metadata templates to support each WP2/3/4 when publishing output data to WP5 for situation assessment, visualization and trust and credibility modelling. An early snapshot of the overall WP5 data model and architecture is also provided for context. |
| **Keyword List:** | Metadata Template, Situation Assessment, Data fusion, Architecture |

# DOCUMENT DESCRIPTION

## Document Revision History

| *Version* | *Date* | *Modifications Introduced* | |
|---|---|---|---|
| | | *Modification Reason* | *Modified by* |
| v0.1 | 18/07/2014 | Initial skeleton document; Feedback from ITINNO team; JSON templates from ITINNO, UKob, CERTH, NCSR'D | ITINNO |
| v0.2 | 05/08/2014 | 1st full draft for internal partner review.; includes all email discussions regarding JSON templates (CERTH, UKob, NCSR'D) | ITINNO |
| v1.0 | 21/08/2014 | 1st release candidate; Includes feedback from CERTH | ITINNO |
| v1.0.1 | 21/08/2014 | Internal Review of report | UKob |
| v1.1 | 29/08/2014 | Version passing internal review | ITINNO |

# CONTENTS

**EXECUTIVE SUMMARY ..... 8**

**1 INTRODUCTION ..... 9**

1.1 SCOPE OF WP5 AND D5.1 ..... 9
1.2 PROBLEM STATEMENT ..... 9
1.3 OVERALL APPROACH ..... 10

**2 ARCHITECTURE FOR METADATA DRIVEN SITUATION ASSESSMENT ..... 11**

2.1 OVERVIEW OF ARCHITECTURE ..... 11
2.2 TECHNOLOGY VIEWPOINT ..... 12
2.3 STORM CONCEPTS ..... 14
2.4 INFORMATION VIEWPOINT ..... 17
2.5 ENGINEERING VIEWPOINT ..... 19

**3 METADATA TEMPLATE DEFINITIONS ..... 26**

**4 CONCLUSIONS ..... 38**

**5 REFERENCES ..... 39**

# LIST OF FIGURES

FIGURE 1: COMPONENT LEVEL WP5 ARCHITECTURE .....11
FIGURE 2: INFORMATION MODEL FOR WP5 .....18
FIGURE 3: LEGEND FOR ENGINEERING VIEWPOINT DIAGRAMS .....19
FIGURE 4: STORM CONTROLLER, PRE-PROCESSING AND PREPARE JSON STORM TOPOLOGIES .....20
FIGURE 5: GEOPARSE AND AGGREGATE GEOPARSED LOCATIONS TOPOLOGY .....22
FIGURE 6: GEO-CLASSIFICATION TOPOLOGY .....22
FIGURE 7: GEOSPATIAL AND TOPIC MODEL TOPOLOGIES .....23
FIGURE 8: MULTIMEDIA PROCESSING TOPOLOGIES .....23
FIGURE 9: SITUATION ASSESSMENT TOPOLOGY .....24
FIGURE 10: EXTERNAL AND TRUST MODELLING TOPOLOGIES .....25

# DEFINITIONS, ACRONYMS AND ABBREVIATIONS

| Acronym | Title |
|---|---|
| AMQP | Advanced Message Queuing Protocol |
| API | Application Programming Interface |
| GPL | Open Source GNU Public License |
| JAR | Java Archive |
| JSON | JavaScript Object Notation |
| JVM | Java Virtual Machine |
| MIT | Massachusetts Institute of Technology<br>In this document used to describe the open source MIT license |
| OpenGIS | Open Geodata Interoperability Specification<br>In this document used to mean OGC's OpenGIS "Simple Features Specification for SQL" standard supported by databases such as PostGIS |
| OSM | Open Street Maps |
| REST | Representational state transfer |
| SPARQL | SPARQL Protocol and RDF Query Language |
| URL | Uniform Resource Locator |
| WMS | Web Mapping Service |
| WP1 | Users Requirements and Regulatory Framework |
| WP2 | Contributor Modelling |
| WP3 | Multimedia Analysis and Indexing |
| WP4 | Context-centric Information Analysis |
| WP5 | Modalities Analysis Framework |

| | |
|---|---|
| WP6 | Framework Integration and REVEAL Applications |

# Executive Summary

The WP5 'modalities analysis framework' is designed to create a situation assessment data fusion framework capable of combining all relevant social, content and context reports from WP2/3/4. This situation assessment picture is expected to evolve in real-time and a set of visualization tools will be created to allow end user analysts to examine interactively the 'big picture'. The situation assessment will also provide evidence for the trust and credibility modelling components in WP5, extracting higher level modalities such as 'truth' and 'credibility'.

This deliverable outlines an architectural framework and information model for the WP5 'modalities analysis framework'. This architecture is based on a combination of Storm controlled processing components, HTTP endpoints, a RabbitMQ message bus for WP2/3/4 to report annotations, situation assessment components to aggregate real-time annotations providing 'evidence' and a knowledge-based approach to trust and credibility modelling. Annotations are self-describing and follow a set of JSON templates defined in this report.

The deliverable D4.1 also contains information on architecture as both WP4 and WP5 processing is based on Storm. The D4.1 architecture viewpoint is a subset of the WP5 viewpoint and as such D5.1 contains a full set of JSON annotations and a full set of defined Storm topologies.

A reader should bear in mind that this deliverable represents a 'snapshot' of work in progress, and the details of JSON annotations and storm topologies will change as the REVEAL project consortium gains valuable practical experience over the project lifetime.

# 1 Introduction

## 1.1 Scope of WP5 and D5.1

The WP5 'modalities analysis framework' is designed to create a situation assessment data fusion framework capable of combining all relevant social, content and context reports from WP2/3/4. This situation assessment picture is expected to evolve in real-time and a set of visualization tools will be created to allow end user analysts to examine interactively the 'big picture'. The situation assessment will also provide evidence for the trust and credibility modelling components in WP5, extracting higher level modalities such as 'truth' and 'credibility'.

The work in WP5 will focus on data fusion of incremental reports (i.e. WP2 social, WP3 content and WP4 context annotations of social media content). These reports will build up evidence for the situation assessment 'big picture'. WP5 will not support on-demand reports (e.g. WP2 index creation, WP3 image similarity checking) unless the results can be published as incremental annotations to content items known to the overall situation assessment.

The WP6 'framework integration and REVEAL applications' will create pilot applications for end users to use. This will include control of the processing (e.g. starting a new real-time WP5 situation assessment) and display of result data (e.g. embed visualizations from WP5 into a pilot app). As such the WP5 visualizations are focussed on allowing data analytics for the situation assessments.

This deliverable D5.1 'analytic report of metadata template definitions' contains a set of report templates that WP2/3/4 can use to publish data to be fused into the situation assessment. We have expanded the scope of D5.1 to include the WP5 architecture to provide some context for how these metadata templates will be processed in WP5.

The deliverable D4.1 also contains information on architecture as both WP4 and WP5 processing is based on Storm. The D4.1 architecture viewpoint is a subset of the WP5 viewpoint and as such D5.1 contains a full set of JSON annotations and a full set of defined Storm topologies.

## 1.2 Problem Statement

The challenge of fusing together specialist reports from raw social media content, social network analysis (i.e. WP2), content analysis (i.e. WP3) and context extraction (i.e. WP4) is a multi-faceted one. Reports will arrive asynchronously from both (near) real-time processing components and batch-style processing components. Reports will be incremental, with initial evidence updated over time (e.g. corrections) and new evidence arriving to add to the live situation assessments. It is expected that multiple analytic threads will run simultaneously (e.g. multiple new stories) and might have high throughputs of social media content per thread (e.g. popular news stories which lots of people comment on in social media).

Concrete motivating scenarios for WP5 can be seen in WP1 (both journalistic and enterprise scenarios). For example the breaking news story about the missing flight MH370 has a number of mini-scenarios that highlight the data fusion challenges well. In the WP1 mini-scenario 'Mapping workbench' the breaking news story sees keywords added incrementally as hashtags become adopted by the public over time and additional aspects of the story spin off new discussion threads. The social media crawlers generate real-time streams of JSON content for WP5 to aggregate. New focus areas are added as the suspected direction of the flight changes during the story lifetime, with each area needing a new cluster node to handle the computation load of geo-parsing. WP5 must

aggregate geoparse results and map them. In the WP1 mini-scenario 'Trending workbench' we see hashtags emerge over time as phrases such as '#mh370' gain popularity in tweets and blogs. Reports of trending topic are sent to WP5 for inclusion into the situation assessment. In the mini-scenario 'Picture' we see journalists sending suspicious images for batch-style manipulation detection processing to be applied. When results are ready the content item annotations are incrementally updated to change the credibility rating of the suspicious images. Processing occurs at scale, expanding dynamically with the news story, and many heterogeneous reports incrementally sent to WP5 for data fusion.

## 1.3 Overall Approach

To meet the scalability and performance challenge WP5 supports a distributed architecture based around a scalable event-driven advanced message queue protocol (AMQP) message bus. Each event on the bus represents an annotation, providing new contextual evidence associated with each social media content item, person or group. Some components are written as Storm topologies, exploiting the highly scalable Storm distributed processing architecture to meet the challenge of high throughput and (near) real-time processing. Other components are written as HTTP services supporting batch-style processing of content where processing time is not so critical. In all cases we support an incremental asynchronous annotation model where new contextual evidence is published onto an AMQP message bus.

To meet the challenge of fusing heterogeneous data WP5 supports an incremental 'self-described' JSON template where components from WP2/3/4 can publish incremental annotations of the raw social media JSON content. The WP5 situation assessment components will aggregate this data in an incremental way, collating and cross-checking evidence for input into both the WP5 visualization components and WP5 trust and credibility model components.

# 2 Architecture for metadata driven situation assessment

## 2.1 Overview of architecture

The WP5 architecture is based on an evidential approach to situation assessment and trust & credibility modelling. In order for a situation assessment to be incrementally built up we expect components from WP2/3/4 to publish annotation on raw social media JSON content. Each annotation represents an extracted 'modality' such as proximity, influence etc. These annotations are published onto a RabbitMQ message bus and aggregated using a scalable storm topology deployment. The use of RabbitMQ and Storm provides us with the ability to easily add extra nodes to a computing cluster to handle extra load (e.g. multiple assessment run in parallel, extra social media streams, extra focus areas etc.).

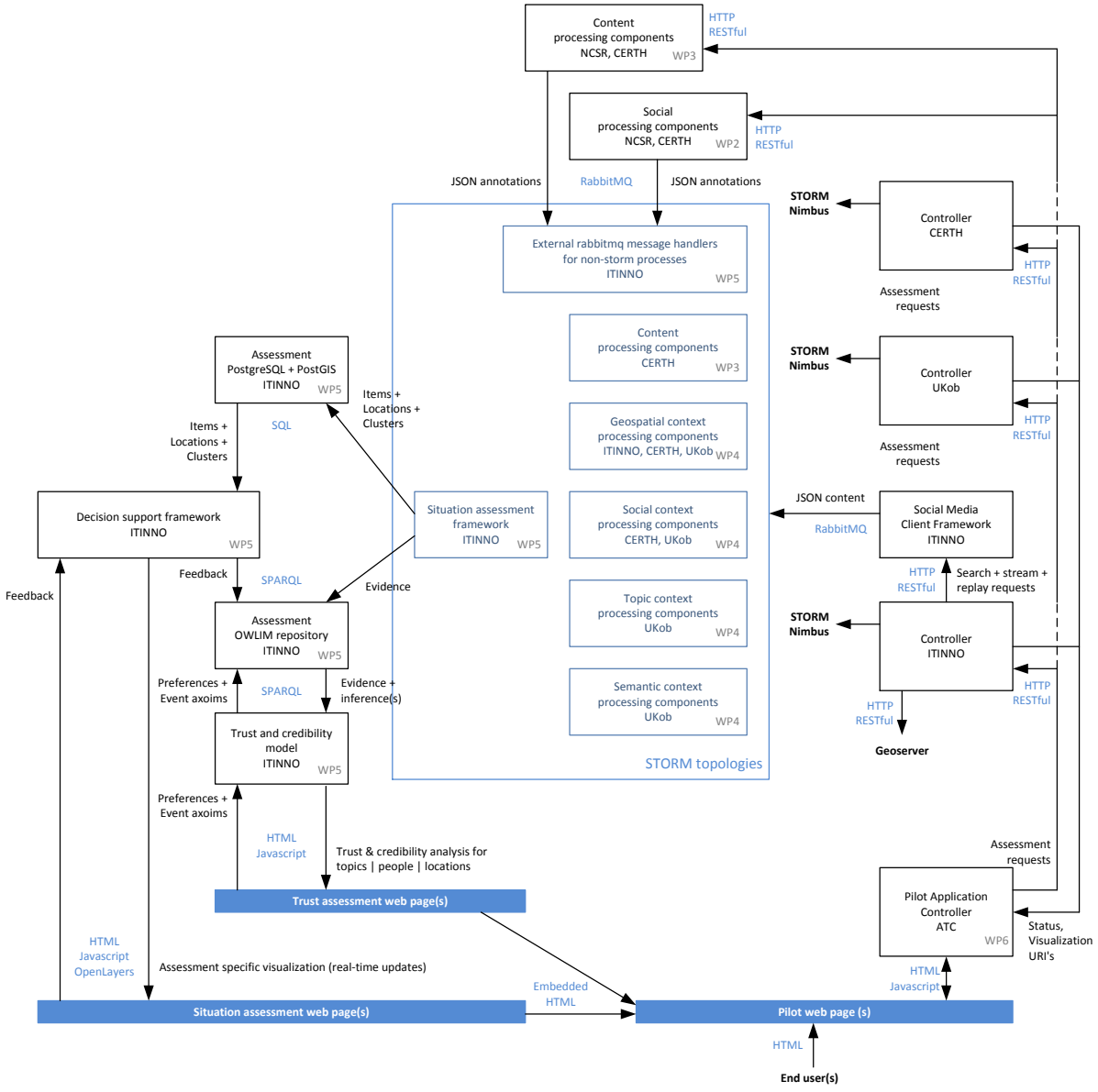


**Figure 1:** Component level WP5 architecture

A high level view of the WP5 architecture is shown in Figure 1. The core of WP5 processing occurs in a storm cluster, with controllers provided by technical partners ITINNO, CERTH and UKob.

The WP5 situation assessment storm topology will maintain a set of master tables for each situation assessment's social media content items. These tables will index the various types of annotations such as matched locations, topics, tags, URL's, contributors etc. New content items will be appended to the content item table, with updated items replacing entries. New annotations will either add new or update entries in the associated annotation table (e.g. table of locations). Statistics of annotation occurrence will be updated on each new item/annotation.

The WP5 visualization component will be based upon a combination of a mapping server (e.g. Geoserver), database access layer and Javascript. Each visualization (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 visualizations are web-based and can be embedded into WP6 pilot applications.

The WP5 trust and credibility model is based around a triple store, following an evidential knowledge-based approach that allows full use of prior knowledge from end users. Each analyst will be able to setup their own personal viewpoint on trust and credibility, as prior knowledge such as lists of trusted authors can be different for each analyst. In addition to the WP5 model there will be localized trust and credibility analysis in WP2/3/4 using low-level information not exposed outside the WP.

## 2.2 Technology viewpoint

RabbitMQ

RabbitMQ [1] is a popular implementation of an event-driven message bus supporting the advanced message queue protocol (AMQP). It supports a broker-based publish/subscribe design pattern. Using an event-driven message bus provides a high performance and scalable messaging system, supporting incremental asynchronous messaging and allowing a loose decoupling of publishers and consumers. It is also possible to setup AMQP brokers in a clustered deployment, providing resilience and fault-tolerance through clustering and mirroring.

The RabbitMQ community has provided brokers for a variety of operating systems and client libraries for most popular programming languages. RabbitMQ ships with an open source Mozilla license.

The design pattern for using RabbitMQ in REVEAL is as a publish/subscribe communication backbone where components looking at modalities can publish incremental and asynchronous updates for WP5 to aggregate into a unified situation assessment. All outputs from WP2/3/4 will be published to a RabbitMQ broker.

Storm

Storm [2] is a distributed real-time computation system. Similar to how Hadoop provides a set of general primitives for doing batch processing, Storm provides a set of general primitives for doing real-time computation. Storm topologies are analogous to a Map Reduce job, with the key difference that a Map Reduce job eventually finishes, whereas a topology runs forever (or until you kill it, of course).

The storm framework can add scalability (e.g. seamless parallel deployment over a cluster) and reliability (e.g. checking for dead processes and resending unprocessed messages) to a real-time processing system. As such it is a good choice for the (near) real-time incremental processing type components we have in REVEAL.

Storm is simple to use, supports many programming languages (e.g. Java, Python, Ruby, Fancy), and is used by many companies including Twitter. It can be deployed on Linux and Windows operating systems. Messaging is based on JSON serialized tuples and various messaging layers are supported (e.g. RabbitMQ, Kestrel, Kafka). There is an active community behind storm releasing updates regularly. Storm ships with an open source MIT license.

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).

An introduction to Storm concepts is provided in section 2.3.

HTTP RESTful Endpoints

In recent years HTTP [3] endpoints have become the defacto standard in the social media space, with all sites (e.g. Twitter, YouTube ...) providing a HTTP API for access to their services. Often the HTTP endpoints follow the representational state transfer (REST) [4] design pattern. REST emphasizes things like separation of concerns and layers, statelessness, and caching, which are common in many distributed architectures because of the benefits they provide. These benefits include interoperability, independent evolution, interception, improved scalability, efficiency, and overall performance. The REST approach focusses on how to manipulate resources through representations, and how to include metadata that make messages self-describing. The HTTP standard and REST approach is now mature and offers a very 'web friendly' way to control services.

The design pattern for HTTP REST in REVEAL is to provide HTTP RESTful endpoints for batch-style processing components, allowing pilot applications to start and stop processing in the context of new situation assessments. Typically documents will be 'dropped off' for processing and 'picked up' when ready. Where useful (e.g. image manipulation detection results on Instagram images) processing results will also be published to the RabbitMQ bus for inclusion into the incremental situation assessment.

Social Media Crawlers

In order to obtain access to raw social media content and peoples profiles a set of social media crawlers will be deployed within REVEAL. This is now standard technology and in REVEAL we will provide HTTP endpoints to control these crawlers, allowing pilot applications to specify things like keywords associated with new situation assessments. For incremental processing crawled social media content (JSON formatted) will be published to the RabbitMQ broker for further processing. For batch-style processing social media content will be published to databases (e.g. MongoDB) where indexing and whole-corpus analysis can be performed.

Incremental JSON annotations

We have adopted an incremental annotation approach since we expect WP2/3/4 annotations to arrive continually and irregularly from (near) real-time and batch-style processing components. The

number of different processing types and processing speeds in REVEAL means a synchronous approach to annotation would be very difficult to manage.

All social media sites support the now defacto standard JSON format, so we have adopted it for all WP5 annotations. It is very easy to incrementally add annotations to a JSON object, simply by adding new keys. We have designed a set of namespaced keys to ensure REVEAL annotations do not clash with existing keys from social media sites.

Distributed database deployments

We expect the data volumes in REVEAL to be large. We will see real-time crawling from various social media sites, resulting in terabytes of data if left running 24/7 over periods of a month or more. We also need access to large image archives (e.g. indexed training images) and geospatial databases (e.g. planet Open Street Map database).

Each WP2/3/4/5 processing component will consider carefully the location of its database(s) of resources. We will use techniques from information retrieval and database theory to decide on table structures, index types and location of databases in a cluster. For work with large data, for example the WP4 geoparse work which needs access to a local planet Open Street Map database, we will avoid transfers of large amounts of data by locating databases locally to the processing node. For less demanding data volumes, such as aggregated data tables in WP5, we can use a central remote database or triple store.

Geoserver

The geospatial visualization will be achieved using an off-the-shelf open source map server such as Geoserver. This approach allows OpenGIS geometry information to be written directly to a PostgreSQL database, and open layers setup (e.g. as a WMS layer) to render this geometry on a map base layer (e.g. Google Maps). The rendering is enacted using a web page, hosted by the decision support framework. Geoserver ships with an open source GPL v2.0 license.

Working with EU Data Protection Law

This is an on-going task in conjunction to guidance from WP1.

We are planning to support a user's right to 'ask for removal' from an information system via a 'blacklist' of author names. This will allow social media crawlers to ignore content found from blacklisted authors.

It is not practical to remove authors from already processed forms of data. For example clusters cannot be 'unpicked' without the re-calculation of the clusters and associated indexes. Therefore our attention is focussed on removal of users and their content at the crawling stage.

The application of blacklists at the visualization stage will also be examined to see if it is practical to block user content prior to visualization. It might be practical to stop user details appearing even if clusters of aggregated content cannot be stopped.

## 2.3 Storm Concepts

This section outlines some basic Storm concepts. The text of this section is a summarized reproduction of text found on the Storm Wiki [7]. It is reproduced here to give a reader essential information about Storm that is needed to later understand the Storm-based architecture.

Topologies

The logic for a real-time application is packaged into a Storm topology. A topology is a graph of spouts and bolts that are connected with stream groupings. A Storm topology is analogous to a Map Reduce job in terms of process flow. One key difference is that a Map Reduce job eventually finishes, whereas a topology runs forever.

Streams

The stream is the core abstraction in Storm. A stream is an unbounded sequence of tuples that is processed and created in parallel in a distributed fashion.

Streams are defined with a schema that names the fields in the stream's tuples. By default, tuples can contain integers, longs, shorts, bytes, strings, doubles, floats, booleans, and byte arrays (it is also possible to define your own serialisers). Every stream is given an id when declared.

Spouts

A spout is a source of streams in a topology.

Generally spouts will read tuples from an external source and emit them into the topology (e.g. a Kestrel queue or the Twitter API). Spouts can either be reliable or unreliable. A reliable spout is capable of replaying a tuple if it failed to be processed by Storm, whereas an unreliable spout forgets about the tuple as soon as it is emitted. Spouts can emit more than one stream.

Bolts

All processing in topologies is done in bolts.

Bolts can do anything from filtering, functions, aggregations, joins, talking to databases, and more. Bolts can do simple stream transformations. Doing complex stream transformations often requires multiple steps and thus multiple bolts. Bolts can emit more than one stream. It is perfectly fine to launch new threads in bolts that do processing asynchronously.

For example, transforming a stream of tweets into a stream of trending images requires at least two steps: a bolt to do a rolling count of retweets for each image, and one or more bolts to stream out the top X images (you can do this particular stream transformation in a more scalable way with three bolts than with two).

When you declare a bolt's input streams, you always subscribe to specific streams of another component; these are explicitly defined for every single stream. For every tuple Bolts process Storm knows when tuples are completed and can eventually determine that it is safe to acknowledge the original spout tuples.

Stream groupings

Part of defining a topology is specifying for each bolt which streams it should receive as input. A stream grouping defines how that stream should be partitioned among the bolt's tasks.

There are seven built-in stream groupings in Storm:

- **Shuffle grouping** - tuples are randomly distributed across the bolt's tasks in a way such that each bolt is guaranteed to get an equal number of tuples.

- **Fields grouping** - the stream is partitioned by the fields specified in the grouping. For example, if the stream is grouped by the "user-id" field, tuples with the same "user-id" will always go to the same task, but tuples with different "user-id"'s may go to different tasks.
- **All grouping** - the stream is replicated across all the bolt's tasks.
- **Global grouping** - the entire stream goes to a single one of the bolt's tasks. Specifically, it goes to the task with the lowest id.
- **None grouping** - this grouping specifies that you do not care how the stream is grouped. Currently, none groupings are equivalent to shuffle groupings. Eventually though, Storm will push down bolts with none groupings to execute in the same thread as the bolt or spout they subscribe from (when possible).
- **Direct grouping** - a stream grouped this way means that the producer of the tuple decides which task of the consumer will receive this tuple. Direct groupings can only be declared on streams that have been declared as direct streams.
- **Local or shuffle grouping** - If the target bolt has one or more tasks in the same worker process, tuples will be shuffled to just those in-process tasks. Otherwise, this acts like a normal shuffle grouping.

Reliability

Storm guarantees that every spout tuple will be fully processed by the topology. It does this by tracking the tree of tuples triggered by every spout tuple and determining when that tree of tuples has been successfully completed. Every topology has a "message timeout" associated with it. If Storm fails to detect that a spout tuple has been completed within that timeout, then it fails the tuple and replays it later.

Tasks

Each spout or bolt executes as many tasks across the cluster. Each task corresponds to one thread of execution, and stream groupings define how to send tuples from one set of tasks to another set of tasks.

Workers

Topologies execute across one or more worker processes. Each worker process is a physical JVM and executes a subset of all the tasks for the topology. Storm tries to spread the tasks evenly across all the workers (need to consider it for load balancing).

For example, if the combined parallelism of the topology is 300 and 50 workers are allocated, then each worker will execute 6 tasks (as threads within the worker).

Storm Cluster

Something similar to Hadoop clusters, but while Hadoop cluster runs a map-reduce, Storm runs topologies. As mentioned earlier, one of the primary differences between map-reduce jobs and topologies is that map-reduce jobs eventually end, while topologies are destined to run until you explicitly kill them. Storm clusters define two types of nodes:

- **Master node (nimbus)** - runs as a daemon process called Nimbus – responsible for distributing code across cluster, assigning tasks to machines, and monitoring the success/failure of units of work.
- **Worker node (supervisor)** - daemon process, called Supervisor – responsible for listening for work assignments for its machine. It then sequentially start and stop worker processes.

Each worker process executes a subset of a topology, so that the execution of a topology is spread across a multitude of worker processes running on a multitude of machines.

Storm ZooKeeper

Sitting between Nimbus and the various Supervisors is the Apache open source project software ZooKeeper. The goal of zookeeper is to enable highly reliable distributed coordination, mainly by acting as a centralised service for distributed cluster functionality.

Storm topologies are deployed to the Nimbus and then the Nimbus deploys spouts and bolts to Supervisors. When it comes time to execute spouts and bolts, the Nimbus communicates with the Supervisors by passing messages to Zookeepers.

Zookeepers maintain all state for the topology, which allows the Nimbus and Supervisors to be fail-fast and stateless: if the Nimbus or Supervisor processes go down then the state of processing is not lost; work is reassigned to another Supervisor and processing continues. Then, when the Nimbus or a Supervisor is restarted, they simply re-join the cluster and their capacity is added to the cluster

Storm UI

The storm user interface is web based and provides visual Storm topology management (e.g. start, stop, some reconfiguration etc.). Storm UI also provides information about errors happening in tasks and fine-grained stats on the throughput and latency performance of each component of each running topology.

Useful references:

- Main Storm webpage [5]
- Main Storm library download page [6]
- Storm documentation [7]
- Storm FAQ [8]
- Storm examples [9]
- Storm java API [10]
- Storm multi-language API [11]

## 2.4 Information viewpoint

The data model used in WP5 is based on an incrementally updated situation assessment model. We support three types of situation assessment - geospatial, social and topical. Figure 2 shows the information model at a high conceptual level. The raw information comes from social media content items and authors. The processing of WP2/3/4 provides annotations to the raw information at higher levels of abstraction (e.g. location, topic, interactions, forensic image analysis results etc.). These annotations are then aggregated to form evidence for the three types of situation assessment, and in turn used to provide evidence for the trust and credibility model. The WP2/3/4 annotations can be cross-checked (e.g. adding credibility if there are multiple reports of a location) and assessed to provide insights into concepts such as proximity (e.g. linked data to nearby locations) and coherence (e.g. geoparse OSM tags matching image feature OSM tags). At the highest level the trust and credibility model provides insights into trust and credibility.

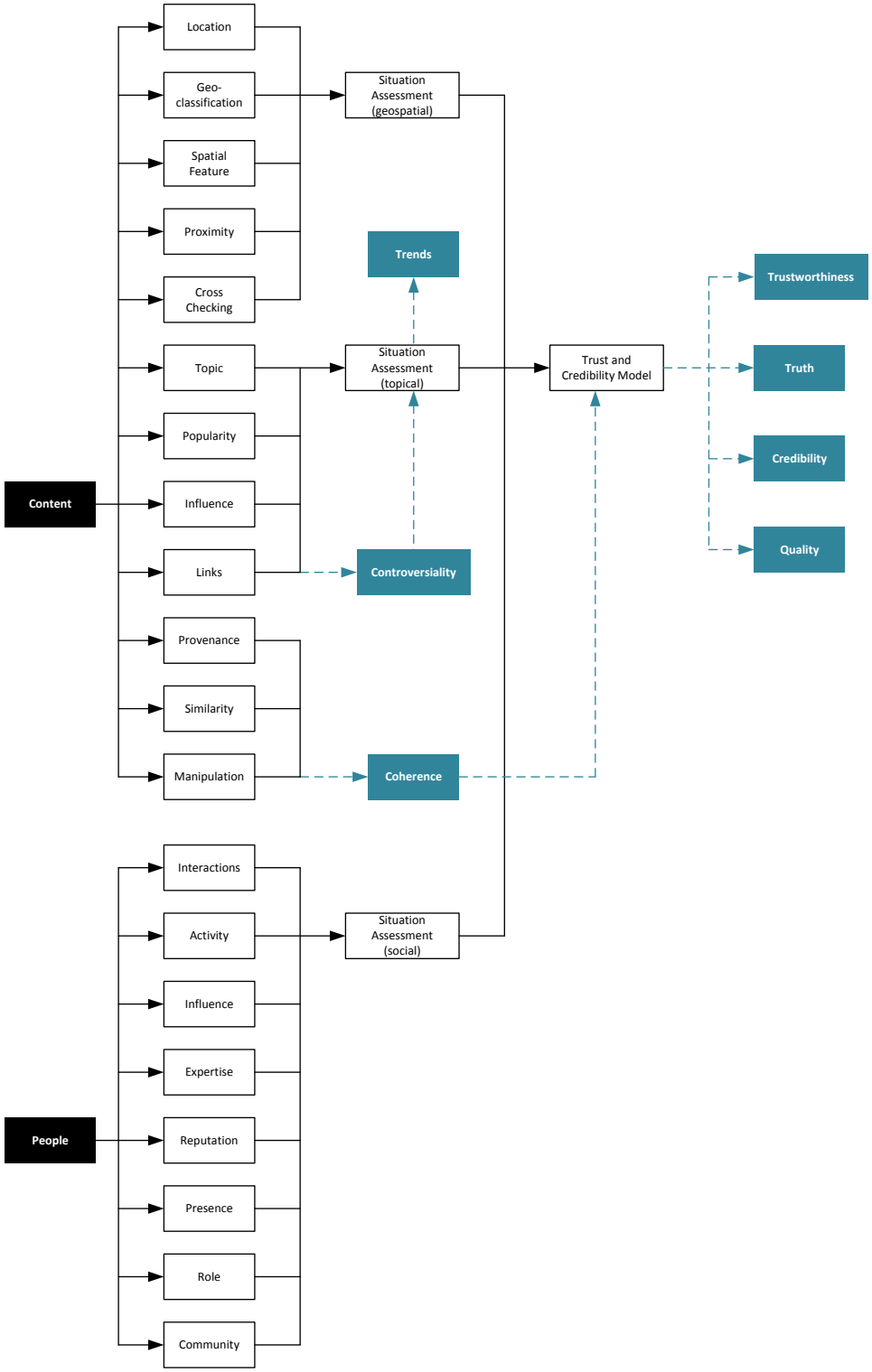


**Figure 2:** Information model for WP5

The approach to trust and credibility modelling is one of an evidential knowledge-based approach. We will incrementally add evidence, from WP2/3/4 and the situation assessments in WP5, to a triple store and allow end user analysts to assert prior knowledge about a news story / enterprise event (e.g. a list of trusted authors). Simple and scalable inference rules, created interactively using a-priori knowledge from end users, will then be applied to classify evidence into groups relevant to each

analyst. Analysts will then engage in an interactive session where SPARQL queries can be postulated over the information model to examine different intersections/unions of classified evidence with the aim of finding clusters of evidence to support an end user hypothesis.

This evidential approach to trust and credibility modelling will be developed further over the next 2 years.

## 2.5 Engineering viewpoint

This section contains storm topology design information to provide an engineering viewpoint on WP5. We intend this section as a 'snapshot' of work in progress, and so provide only a brief description of the most mature topologies at this stage of the project. A complete set of storm topologies is available as an internal consortium document, and forms part of the agile development and integration process, to be updated over the lifetime of the project.

Figure 2 shows the legend for all engineering diagrams in this section. Details of the techniques used in WP2/3/4 topologies can be found in D2.1, D3.1 and D4.1.

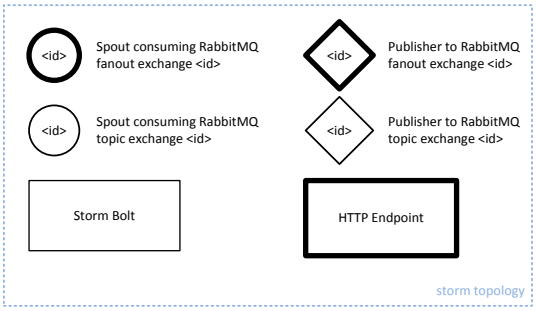


**Figure 3:** Legend for engineering viewpoint diagrams

Storm controller, pre-processing and prepare JSON storm topologies

The storm controller and social media client framework are both HTTP RESTful endpoints (API detailed later in this section). The 'pre-processing' topology takes new focus area requests and queries locations from the planet OSM database. New locations are then allocated to a location map and geoparse topologies spawned. In this way the number of geoparse topologies can be dynamically tailored to the scale of focus areas (e.g. one city, the entire coastline of a country). The 'prepare JSON' topology parses all raw JSON content and adds itinno namespaced annotations. This provides a unique identifier, for use later in aggregation of annotations, and a set of common keys that all processing components can use (e.g. itinno:text).

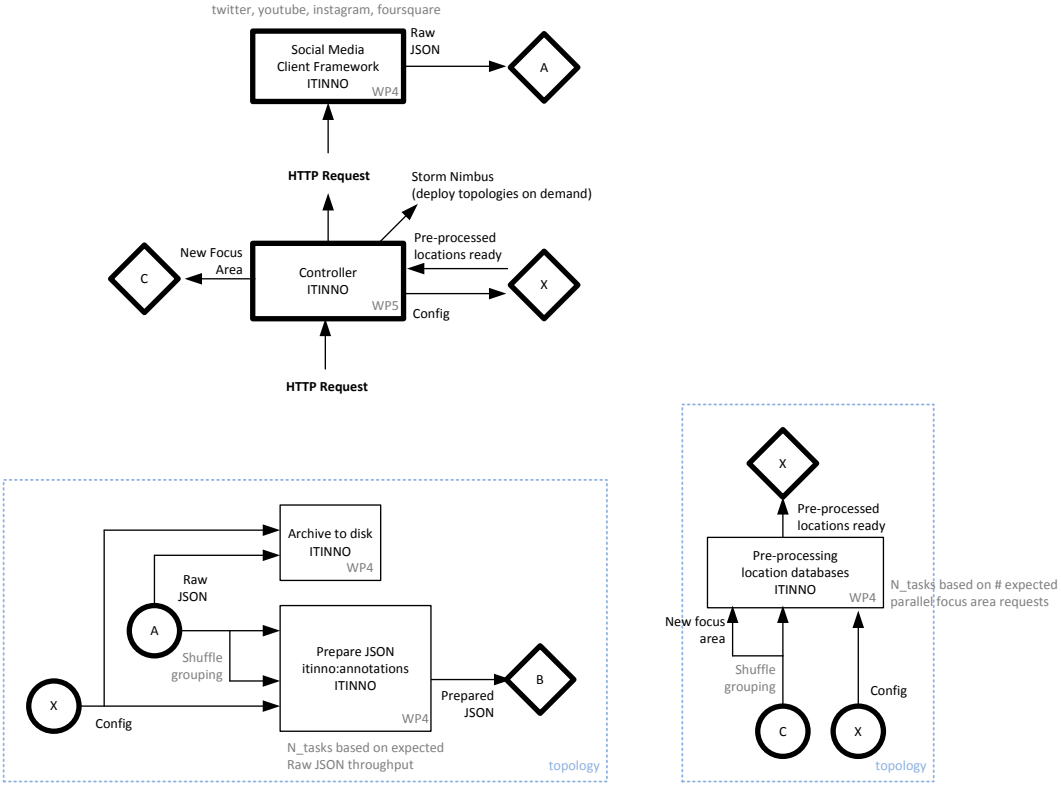


**Figure 4:** Storm controller, pre-processing and prepare JSON storm topologies

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: UKob 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>
```

HTTP endpoint: Social media client framework

```
GET /
GET /<media>
GET /<media>/<id>
POST /<media>/search
POST /<media>/stream
POST /<media>/replay
 --> replay a set of JSON objects serialized on disk
DELETE /<media>/stream/<id>
DELETE /<media>/replay/<id>
```

Geoparse and aggregate geoparsed locations topology

The 'geoparse' topology caches into memory sets of locations for real-time matching. It then matches these locations to real-time social media content and provides a location annotation. Since location sets are deliberately fragmented, to parallelize the location matching process, an 'aggregate geoparsed locations' topology is also provided to give a single annotation set for all known locations.

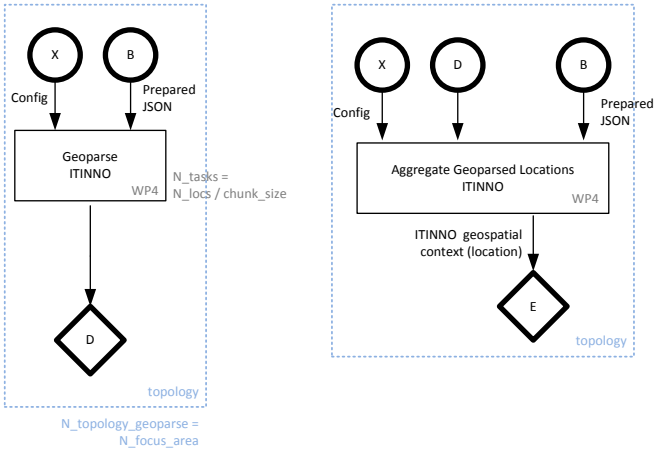


**Figure 5:** Geoparse and aggregate geoparsed locations topology

Geo-classification topology

The 'geo-classification' topology will classify textual content based on the way location mentions are referred to. The results of the geo-classification storm topology will be, for each content item, a set of classes and probabilities that the text matches these classes. Possible class sets are past/present/future reports (e.g. 'Boston was bombed last year'), in-situ/remote reports (e.g. 'I saw on the TV that Boston was bombed') and confirmation/denial (e.g. 'Boston was bombed - it's not a hoax').

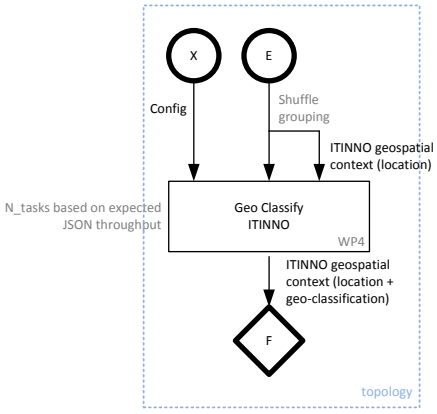


**Figure 6:** Geo-classification topology

Geospatial and topic model topologies

The 'topic model' topology identifies occurrences of topics in text using a common topic model. Each social media content item is annotated with a set of topics and a confidence rating of the match. The topic annotations are passed to a 'Focused Geospatial Exploration' topology where linked data annotations are added (e.g. semantically relevant locations to this content item).

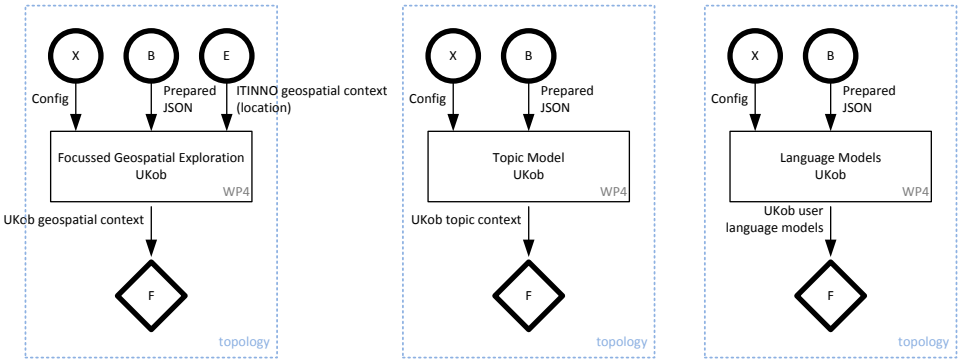


**Figure 7:** Geospatial and topic model topologies

Multimedia processing topologies

The 'multimedia processing' topologies add annotations based on image geolocation and image feature extraction. Image geo-location compares images to a locally indexed training image database looking for matches. Image feature extraction looks for high level image features associated with OSM tags, such as forests, urban, coastline etc.

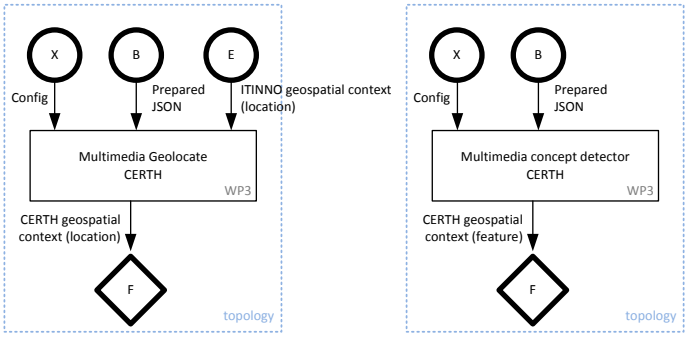


**Figure 8:** Multimedia processing topologies

Situation assessment topology

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.

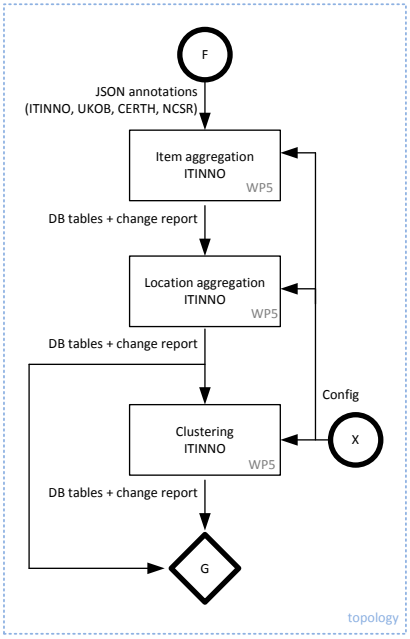


**Figure 9:** Situation assessment topology

External and trust modelling topologies

A number of REVEAL components need a batch-style processing and are not well suited to running as storm topologies. As such we provide a topology to aggregate reports from non-storm components. There are also storm topologies to look at behaviour reports (e.g. four square checkins) and add evidence to a triple store for access by the knowledge-based WP5 trust and credibility model.

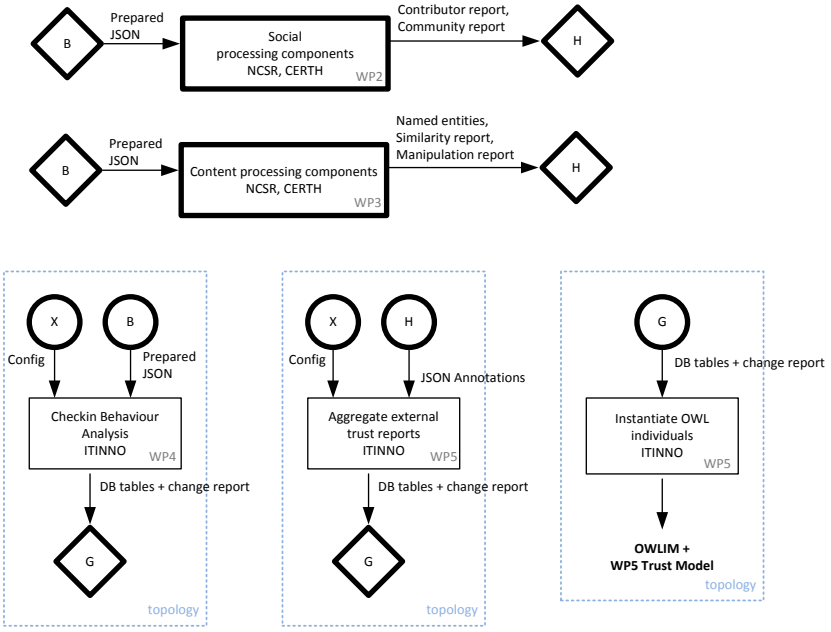


**Figure 10:** External and trust modelling topologies

# 3 Metadata template definitions

Information published to WP5 is in the form of JSON formatted annotations to the original JSON social media content item, contributor or group of contributors. This allows 'self-describing' metadata to be added to the raw original content. Annotations can be incrementally added and the data fusion components do not need to wait for all annotations to arrive at the same time (i.e. we will support an incremental aggregation algorithm).

This section contains details of the known JSON annotations defined so far for input to WP5. We expect JSON templates to be updated during the project to reflect experiences gained in WP2/3/4 regarding what information can be produced and what information is of most value in the context of the use cases.

Raw JSON

JSON object that could any one of the following (more will be added over time):

- JSON tweet
- JSON You Tube item
- JSON Instagram data
- Four Square JSON venue

Prepared JSON

Original raw JSON object with the addition of 'itinno' namespaced key/value pairs. This allows processing components to use keys such as itinno:text or itinno:tokens, and not worry if the content item is a Tweet, You Tube video or Instagram picture. The original raw JSON is still available for social media specific processing actions.

Prepared JSON
+ Raw JSON
+ itinno:item_id = unique ITINNO item_id for content item (used for aggregation)
+ itinno:type = type of social media item
e.g. tweet | youtube | instagram | foursquare
+ itinno:title = title text (if any)
+ itinno:description = body text (if any)
+ itinno:comment = concatenated comment text for item (if any)
+ itinno:text = concatenated text from title, body and comments
+ itinno:tokens = Ngram tokenized text (ITINNO choice of tokenizer)
+ itinno:source_id = site specific id to original content
+ itinno:source_uri = URI to original content
e.g. https://twitter.com/stuart_e_middle/status/426317929936715776
+ itinno:timestamp = ISO formatted creation time
e.g. 2014-01-10T00:00:05+00:00
+ itinno:geotag = OpenGIS = geotag (if it has one)
e.g. POINT(X Y)
+ itinno:tags = [tags] = list of hashtags / tags / keywords present in content
+ itinno:urls = [urls] = list of expanded urls present in text
+ itinno:connected_count = int = # of times retweeted / commented / liked
+ itinno:liked_count = int = # of times marked as favourite / bookmarked / liked
+ itinno:original = bool = false if this is a retweet or comment
+ itinno:lang = BCP 47 two character language code (see http://tools.ietf.org/html/bcp47)
e.g. en, de
+ itinno:author_id = site specific id of content author
+ itinno:author_uri = URI to content author
e.g. https://twitter.com/stuart_e_middle

ITINNO JSON geospatial context (location)

Prepared JSON object with the addition of itinno geoparse annotations.

```
ITINNO JSON geospatial context
  + Prepared JSON
  + itinno:loc_set = <loc_info>

<loc_info> = [<source_loc_id>, <token_start>, <token_end>, <geom>, [<osm_id>]*N_entity,
[<linkeddata_uri]*N_entity, {<osm_tag>:<value>}*N_tags, [<osm_id>]*N_admin_parents,
<score>, [<multi_lang_name>,...]*N_langs  ]*N_locs

<geom> = OpenGIS geometry
   e.g. POINT(51.123 -1.900)
<linkeddata_uri = http://linkedgeodata.org/page/triplify/[node|way|relation]<odm_id>
<osm_tag> = osm tag name - see http://wiki.openstreetmap.org/wiki/Map_Features
<score> = int = 1 + number of admin regions present in text for this location [measure of
confidence - higher is better]

note: scores are for relative ranking and filtering. they are not normalized values that can
be meaningfully compared with other types of score.
```

ITINNO geospatial context (location + geo-classification)

JSON object containing itinno geoparse and classification annotations.

```
ITINNO geospatial context
  + itinno:item_id = unique ITINNO item_id for content item
  + itinno:loc_set = <loc_info>
  + itinno:class_set = { <class> : <score> } * N_class

<class> = class label
     e.g. past | present | future | none
     e.g. first person | third party
<score> = float = 0..1 = probability returned by classifier

note: scores are for relative ranking and filtering. they are not normalized values that can
be meaningfully compared with other types of score.
```

CERTH geospatial context (location)

JSON object containing CERTH multimedia geolocation annotations.

```
CERTH geospatial context (location)
 + itinno:item_id = unique ITINNO item_id for content item
 + certh:loc_set = [ source_loc_id, loc_name, <geom>, <score> ] * N_geolocation

<score> = float = 0..1 = geolocation confidence
<geom> = OpenGIS geometry
   e.g. POINT(51.123 -1.900)

note: normalized scores can be used for data fusion. other types of scores can be used for
relative ranking and filtering.
```

CERTH geospatial context (feature)

JSON object containing CERTH multimedia concept detection annotations.

```
CERTH geospatial context (feature)
 + itinno:item_id = unique ITINNO item_id for content item
 + certh:concept_set = [
    {<tag>:<value>}*N_osm_tags,
    [<tag>]*N_caltech_tags,
    [<tag>]*N_nuswide_tags,
    <concept_score> ]*N_concept

<value> = value specializing the OSM tag category
<concept_score> = float = 0..1 = concept identification confidence value

note: OSM tags also have values that specialize them (e.g. {natural:beach},
{shop:supermarket} ), whereas CALTECH and NUSWIDE use a simple list of categories

note: normalized scores can be used for data fusion. other types of scores can be used for
relative ranking and filtering.
```

For OSM tag category the following feature to OSM tag mappings will be initially used. These might be revised based on evaluation against datasets gathered within the project.

| CALTECH feature | OSM tag |
|---|---|
| Industrial | {plant : *} |
| Industrial | {landuse : industrial} |
| MITcoast | {natural : costline} |
| | {natural : beach} |
| | {place : island} |
| | {leisure : beach_resort} |
| CALsuburb | {place : suburb} |
| MITforest | {natural : wood} |
| | {landuse : forest} |
| MIThighway | {highway : *} |
| MITinsidecity | {place : city} |
| MITmountain | {natural : arete} |
| | {natural : peak} |
| | {natural : ridge} |
| NUS-WIDE feature | OSM tag |
| airport | {aeroway : *} |
| bridge | {building : bridge} |
| bridge | {bridge : yes} |
| castle | {historic : castle} |
| boats | {leisure : slipway} |
| | {leisure : marina} |
| glacier | {natural : glacier} |
| lake | {natural : water} |
| railroad | {railway : *} |
| | {route : railway} |
| sand | {surface : sand} |
| | {natural : costline} |
| | {natural : beach} |
| | {place : island} |
| | {leisure : beach_resort} |
| tower | {man_made : tower} |
| road | {highway : *} |
| | {route : road} |
| plants | {landuse : greenfield} |

CERTH topic context

JSON object containing CERTH tag based language models.

```
CERTH topic context
  + itinno:item_id = unique ITINNO item_id for content item
  + certh:tag_set = [tag_id, tag_name, <relevance_score>] * N_tags

<score> = float = 0..1 = geolocation confidence
<relevance_score> = float = 0..1 = tag relevance value
```

CERTH social context for content

JSON object containing CERTH social context annotations. The social media context describes a cluster of activity (e.g. likes, upvotes ...) around a single content item.

CERTH social context
+ itinno:item_id = unique ITINNO item_id for content item
+ certh:context_id = unique context cluster id containing this items activity
+ certh:implicated_user_count = number of unique users interacting with content
+ certh:implicated_user_set = [ itinno:author_uri ] * N_users
+ certh:upvote_count = number of upvotes for content
+ certh:popularity = content popularity is a simple score/measure based on implicated user and upvote counts
+ certh:comment_count = count of content item comments
+ certh:diffusion_complexity = float = measure that characterizes the complexity of the diffusion tree
+ certh:influence = score of this content item influence
+ certh:user_recurrence_index = float = 0..1 = measure that characterizes user recurrence to a discussion
+ certh:effective_contributor_count = float = measure that approximates the active contributors in the discussion, in contrast to one-time contributors
+ certh:discussion_complexity = float = measure that characterizes the complexity of the discussion tree
+ certh:community_cohesion = float = 0..1 = measure that characterizes the density/cohesion of the implicated user underlying network
+ certh:controversiality = score of how controversial a content item is
+ certh:temporal_statistics = [ start_time, end_time, <diffusion_complexity_rate>, <discussion_complexity_rate>, <popularity_trend>, <influence_trend>, <controversiality_trend> ]

<diffusion_complexity_rate>= float = rate of change of diffusion complexity in a pre-defined time frame, e.g. 24 hrs
<discussion_complexity_rate>= float = rate of change of discussion complexity in a pre-defined time frame, e.g. 24 hrs
<popularity_trend>= float = 0..1 = normalized score of popularity trend prediction in a pre-defined time frame, e.g. 24 hrs
<influence_trend>= float = 0..1 = normalized score of influence trend prediction in a pre-defined time frame, e.g. 24 hrs
<controversiality_trend>= float = 0..1 = normalized score of controversiality trend prediction in a pre-defined time frame, e.g. 24 hrs

CERTH similarity report

JSON object containing CERTH similarity report annotations. The similarity report describes if an image is similar to other images in the indexed training set.

```
CERTH similarity report
+ certh:source_uri (e.g. instagram URI)
+ certh:created_date
+ certh:image_set=[image_id, <created_date>,<similarity_score>]*N_similar_images
+ certh:timedelta_from_earliest_image

<similarity_score>=float=0...1=Similarity score value
```

CERTH manipulation report

JSON object containing CERTH manipulation report annotations. The manipulation report describes if there is evidence that an image has been manipulated in some way.

```
CERTH manipulation report
+ certh:source_uri (e.g. instagram URI)
+ certh:created_date
+ certh:manipulation_set=[<manipulation_id>, <confidence_score>,
<output_parameter_set>]*N_predefined_manipulations
+ certh:manipulation_score = <manipulation_score>

<manipulation_id>=Arbitrary id to identify a manipulation from a predefined list

<confidence_score>=float=0...1=Confidence that the specified manipulation is present

<output_parameter_set>=Additional parameters concerning the specified manipulation,
e.g. bounding box, original image ids in case of splicing etc. Concrete syntax will be defined
once T3.3 has started.

<manipulation_score>=Quantifies the extent to which an image has been manipulated.
```

UKOB social context for users

JSON object containing UKOB social context annotations. The annotations are based on interactions among users as well as between users and content items. This leads to two foreseen types of annotation. The first type relates to the annotation of content items and interactions with them:

UKOB social context based on interaction between users and content items
+ itinno:item_id = unique ITINNO item_id for content item
+ ukob:response_count = integer count of responses (as far as known)
+ ukob:bidir_discussion = Boolean
+ ukob:distance_to_discussion_root = integer depth in the discussion tree

The second type of annotation relates to users and how they interact with each other directly or indirectly:

UKOB social context based on interaction among users
+ itinno:author_id = site specific id of content author
+ ukob:contribution_count = integer; absolute number of contributions
+ ukob:initiated_thread_count = integer; absolute number of started discussions
+ ukob:avg_post_per_thread = float; average number of contributions in a discussion
+ ukob:stddev_post_per_thread = float; standard deviation of the number of contributions in a discussion
+ ukob:indegree_ratio = float; ratio of users replying to this user
+ ukob:post_reply_ratio = float; ratio of posts that receive a reply
+ ukob:thread_initiation_ratio = float; ratio of posts of user starting a thread
+ ukob:bidir_thread_ratio = float; ratio of thread where the user has a bi-directional disucssion
+ ukob:bidir_neighbour_ratio = float; ratio of users that interact with the user in a bi-directional way

UKob geospatial context

JSON object containing UKOB geospatial annotations.

UKob geospatial context
+ itinno:item_id = unique ITINNO item_id for content item
+ ukob:explored_entity_urls = <linkeddata_uri*N

UKob topic context

JSON object containing UKOB topic annotations.

```
UKOB topic context
 + itinno:item_id = unique ITINNO item_id for content item
 + ukob:topic_set = [ topic_id, topic, <score_topic> ] * N_topic

<score_topic> = float = likelihood of topic

note: scores are for relative ranking and filtering. they are not normalized values that can be meaningfully compared with other types of score.
```

UKob language model

JSON object containing UKOB language model annotations

```
UKOB user language models
 + itinno:item_id = unique itinnoitem_id for content item
 + ukob:user_glm_perplexity = float = perplexity of GLM trained for author in explaining the given content item
```

NCSR'D named entities

JSON object representing named entities and interesting clustering of textual entities.

```
NCSR named entities
 + itinno:item_id = unique ITINNO item_id for content item
 + ncsr:named_entity_set = { named_entity : frequency } * N_named_entities
 + ncsr:relation_set = [<text_cluster_id>, <relation>, <text_cluster_id>]
 + ncsr:text_cluster_set = [<text_cluster_id>, { <topic> : topic_id } * N_topic ]

<topic> = UKob topic model label
<relation> = free text phrase connecting text clusters (e.g. 'went to', 'lives at')
```

NCSR'D + CERTH Contributor report

JSON object representing a specific person/group. This annotation provides information on the influence and other attributes this person/group has on the social networks they participate within and recent activity levels. It also provides information regarding the trustworthiness and credibility of this person.

Contributor report JSON
+ ncsr:contributor_id = unique id for disambiguated authors
+ ncsr:members = { itinno:author_uri : itinno:author_id } * N_sites
+ ncsr:home_location = { loc_name : geotag } * N_locations (if multiple)
+ ncsr:influence = [ analysis_start_time, analysis_end_time, <influence> ]
+ ncsr:expertise = [ analysis_start_time, analysis_end_time, <expertise> ]
+ ncsr:reputation = [ analysis_start_time, analysis_end_time, <reputation> ]
+ ncsr:presence= [ analysis_start_time, analysis_end_time, <presence> ]
+ certh:type = [ <user_type>, <score> ] *N_types

<influence> = { <topic> : <score> } * N_topics
<expertise> = { <topic> : <role> } * N_topics
<reputation> = { <topic> : <score> } * N_topics
<presence> = { <topic> : <score> } * N_topics
<topic> = label for topic (e.g. from UKob topic model)
<role> = label for role (e.g. tech_expert, visionary, newbie)
<score> = float = 0..1 (1 = maximum influence, reputation, presence score)
<user_type> = one possible type of user (e.g. person, spam, bot, etc.)

itinno:author_id = site specific id of content author
itinno:author_uri = URI to content author
e.g. https://twitter.com/stuart_e_middle

note: D2.1 describes the WP2 internal data model for contributors and groups.

note: normalized scores can be used for data fusion. other types of scores can be used for relative ranking and filtering.

NCSR'D + CERTH Community report

JSON object representing a group of people. This annotation provides information on the community this person/group has on the social networks based on their activity and network connections. It is incrementally updated over time as new behaviour emerges.

Community report JSON
+ ncsr:group_id = unique id for group / cluster of authors
+ ncsr:members = [ ncsr:contributor_id ] * N_contributors
+ ncsr:analysis_time = timestamp frequency data was calculated
+ ncsr:tag_set = { tag : frequency } * N_tags
+ ncsr:named_entity_set = { named_entity : frequency } * N_named_entities
+ certh:popularity= [<score>]
+ certh:cohesion = [<score>]
+ certh:stability = [<score>]

<score> = float = 0..1 (1 = maximum popularity, cohesion, stability score)

note: all contributor_id's that appear in the community report MUST also appear in the contributor report.

note: D2.1 describes the WP2 internal data model for contributors and groups.

# 4 Conclusions

This deliverable outlines an architectural framework and information model for the WP5 'modalities analysis framework'. This architecture is based on a combination of Storm controlled processing components, HTTP endpoints, a RabbitMQ message bus for WP2/3/4 to report annotations, situation assessment components to aggregate real-time annotations providing 'evidence' and a knowledge-based approach to trust and credibility modelling. Annotations are self-describing and follow a set of JSON templates defined in this report.

The deliverable D4.1 also contains information on architecture as both WP4 and WP5 processing is based on Storm. The D4.1 architecture viewpoint is a subset of the WP5 viewpoint and as such D5.1 contains a full set of JSON annotations and a full set of defined Storm topologies.

A reader should bear in mind that this deliverable represents a 'snapshot' of work in progress, and the details of JSON annotations and storm topologies will change as the REVEAL project consortium gains valuable practical experience over the project lifetime.

# 5 References


[1] RabbitMQ, "Messaging that just works", http://www.rabbitmq.com/

[2] Storm, "Distributed and fault-tolerant realtime computation", https://storm.incubator.apache.org/

[3] Hypertext Transfer Protocol -- HTTP/1.1, W3C, http://www.w3.org/Protocols/rfc2616/rfc2616.html

[4] R.T.Fieldings, "Architectural Styles and the Design of Network-based Software Architectures", University of California, Irvine, 2000.

[5] Storm Wiki: http://storm.incubator.apache.org/

[6] Storm Wiki: library download page: http://storm.incubator.apache.org/downloads.html

[7] Storm Wiki: documentation: http://storm.incubator.apache.org/documentation/Documentation.html

[8] Storm Wiki: FAQ: http://storm.incubator.apache.org/documentation/FAQ.html

[9] Storm Wiki: examples: https://github.com/nathanmarz/storm-starter

[10] Storm Wiki: Java API: http://nathanmarz.github.io/storm/

[11] Storm Wiki: Multi-language API: https://github.com/nathanmarz/storm/wiki/Multilang-protocol