FP7 Project ASAP
Adaptable Scalable Analytics Platform

D6.3 InfoViz Services v2

WP 6 – Information Visualization
webLyzard technology

Nature: Report
Dissemination: Public

Version History

Version	Date	Author	Comments
0.1	06 Jun 2015	W. Rafelsberger	Initial Version based on D6.2
0.2	15 Jun 2016	A. Hubmann	Revision and Search API
0.3	16 Jun 2016	A. Weichselbraun	System Architecture Extension
0.4	20 Jul 2016	A. Scharl	Major Revision
0.5	16 Aug 2016	W. Rafelsberger	GeoMap, Visual Analytics Components
0.6	25 Aug 2016	W. Rafelsberger	VaaS Architecture
1.0	31 Aug 2016	A. Scharl	Revision and Final Edits

Acknowledgement

This project has received funding from the European Union's 7th Framework Programme for research, technological development and demonstration under grant agreement number 619706.

Executive Summary

The main goal of WP6 is the development of high-performance visual interfaces as part of the ASAP system architecture to help analysts navigate big data repositories, in real time and across multiple dimensions (temporal, spatial, etc.). Special emphasis is placed on sub-second response times of the user interface, and the ability to incorporate metadata as additional context information when analyzing complex relations in Web content (WP8) or telecommunications data (WP9).

Research and Development Roadmap

Rendering complex, manifold relations and patterns in contextual information spaces (Scharl et al., 2015) requires an integration of metadata extracted from text documents (unstructured) with statistical indicator data (structured). For this purpose, ASAP developed a number of core components in Year 1 (acquisition, representation and visualization of statistical data; temporal controls) and Year 2 (data layers and geospatial projections).

Year 2 efforts resulted in a first version of an open API for effective data interchange and integrating the visualizations into a wide range of applications. In Year 3, the API has been extended and adapted to support dynamic re-calibration of processing workflows (T5.3), and to provide key functionalities for the Web content analytics and telecommunications use cases (WP8, WP9).

Data-Driven Documents (D3) Standard

To ensure rapid prototyping of the visual methods developed, and a seamless integration into the ASAP Dashboard (T6.3)[1] and external Web-based applications, WP6 uses the D3 JavaScript library (Bostock et al., 2011).[2] D3 is perfectly suited for the purposes of ASAP, since it is not focused on a new grammar for graphics, but rather on integrating existing standards to create effective visualizations.

Significant Results

T6.1 focused on methods to collect, represent and visualize statistical data, to be linked together via the ASAP Dashboard (T6.3). This included a linked data indexer, as well as a set of visualization components for the rapid rendering of complex statistical data (using color coding to show metadata attributes, and interactive mechanisms to select appropriate timescales). Development (T6.2, T6.3) and evaluation (T6.4) efforts in the second half of the ASAP project resulted in the following visualization workflow management and visual analytics components:

[1] asap.weblyzard.com
[2] www.d3js.org

Visualization Workflows

- **Application Programming Interface (API)** for structured and unstructured data, which enables the integration of the WP6 visualization engine with other WPs. Included in Appendix C, this API not only allows rendering multi-source data for the use case applications (WP8, WP9), but also will play an important role in the ASAP Exploitation Plan (T10.2) by supporting a flexible and scalable *Visualization-as-a-Service* (VaaS) approach.
- **Containers** bundle together data acquisition, transformation and visualization functionality; we plan to offer these functions using a "Container-as-a-Service" (CaaS) approach (T10.2).
- **Event and notification system** for intra-module communication in multiple contexts and support for manipulation of intermediary and final data sets.
- Substantial **performance gains** through improved *data structures* for document mapping, *optimized queries*, and a revised *indexing strategy*.

Visual Analytics Components

- **Charting module –** new *layout,* interactive *features* based on feedback from the Y1 and Y2 project reviews, as well as from T6.4 evaluations; consolidation by removing dependencies from third-party libraries.
- **Adaptive tooltips** and **context menus** as a user-friendly way to trigger on-the-fly query refinements.
- **Geographic Map:** (i) *custom base layers* and *data manipulation options*; (ii) incremental versions of statistical data visualizations – circular markers, choropleth; (iii) dynamic clustering to allow adaptive data exploration from high-level views to regional/local data; selected features will be published under an Apache open source license.[3]

Application and Dissemination

- Prototype of the **ASAP Dashboard** (T6.3) with real-time *content feeds* and a *rapid synchronization* of *multiple coordinated views,* each view representing one or more visual analytics components.
- **Best Paper Award** at the *49th Hawaii International Conference on System Sciences*; **Journal articles** in *Semantic Web Journal* (Brasoveanu et. al., 2017) and IEEE Systems (Scharl et al., 2016).

[3] www.github.com/weblyzard/infovyz

Table of Contents

Introduction

The interactive visualizations of WP6 are intended to support free insight generation without prior modelling of a domain, embracing both unstructured (Web intelligence) and structured (linked data) sources. Shneiderman et al. (1996) present a taxonomy of data types in the context of visualization, including temporal and multivariate data.

ASAP will dynamically combine such data types on the fly, taking into account the use case specifics and current user tasks. Time is a particularly important dimension to consider, and was as such a focus of the work conducted in T6.1. The resulting interactive visualizations should (i) reveal complex patterns and evolving trends, (ii) provide flexible mechanism to select appropriate timescales, and (iii) are capable of rendering a considerable amount of data spanning multiple sources and significant timescales – without relying solely on aggregation, which might hide important facts.

Interactive Visualization of Heterogeneous Data from Multiple Sources

Visualization is an effective means to help analysts make sense of the current data deluge. Quite often it is not enough to design a single visualization, but rather a set of visualizations to get a better sense of hidden patterns and trends, as different images might send different signals to the user. If this discovery process is to be effective, visualization components need to be able to use large quantities of data from heterogeneous data sources. A telecommunications analyst who wants to visualize call metadata from various cities, for example, might require additional information besides call metadata (see WP9); e.g.:

- news or social media coverage about the observed cities to correlate peaks in the number of calls with co-occurring events such as music concerts, sports events or political campaigns;
- population statistics, GDP data or statistical indicators from the respective cities, assuming such datasets are available in an RDF or JSON format;
- patterns that correlate call metadata (aggregated and anonymized) with statistical data and/or news and social media coverage.

In order to be able to cope with such requirements, a state-of-the-art visualization engine and dashboard needs to include not just a set of appropriate visual methods, but also components that support:

- the parallel processing of a *wide variety of data types*, including *semantic data types* like geolocation, dates, etc.;
- the remix of data from a wide variety of data sources regardless of *domain*, *structure* (structured or unstructured), or *provenance*;
- the possibility to extract various types of aggregated statistics or the most important entities, and means to select, sort and summarize the data.

Data Matching and Integration

Without an integrated backend that provides such services, any visualization service will not reach the scale and depth needed to create on-the-fly visualizations from heterogeneous data sources. Among the most complex problems that such a backend needs to solve is the *matching of data sources to different visualizations*. Several solutions have been proposed, but there is still no widely accepted standard to flexibly integrate and visualize heterogeneous data sources. In general the problem of resolving the entities from different datasets to the same common real-world entities is known as *data matching* or *record linkage* since the 1960s (Koudas et al., 2006). More recent solutions to address the problem of flexible integration for automated visualization include *schema matching* (Cammarano et al., 2007), *ontology based information extraction and integration* (Buitelaar et al., 2008), *data wrangling* (Kandel et al., 2011) or *proactive wrangling* (Guo et al., 2011) via interactive data transformation scripts, *scalable data curation* (Stonebraker et al., 2013), *human data wrangling* following a *crowdsourcing* approach (Clow, 2014), as well as *visual embedding and visual product spaces* (Demiralp et al., 2014).

In addition to these automated visualization models, a number of models mostly focused on automated visualization of structured data (and especially Linked Data) include *Ontology Based Data Access* (Giese et al., 2013), *Linked Widgets* for exploiting governmental Linked Data (Trinh et al., 2013), *LDVM* or Formal Linked Data Visualization Model (Brunetti et al., 2013), universal data cube *concordance model* (Kelleher, 2014), and *OLAP4LD* – which is a generalization of OLAP for various types of linked data (Harth et al., 2014).

Despite the availability of various models and related tools, the field of data matching is still in its infancy. This also suggests that there is still a lot of work to be done in order to get to a place where automated visualization systems are mature and flexible enough to visualize any type of heterogeneous data source on the fly. *Interoperability*, *speed* and *scalability* are obviously other factors that need to be taken into account when designing such a system.

Visual Dashboard to Explore Contextualized Information Spaces

ASAP will integrate real-time data feeds from multiple sources via an open API (see Appendix C), which allows uploading structured and unstructured datasets, searching for specific sets of indicators or documents, and embedding individual visualizations in Web-based applications, to be rendered in real time (T6.1, T6.2).

The ASAP dashboard (T6.3) combines several visualizations to represent the contextualized information space using a multiple coordinated view approach. Synchronized widgets show the various metadata dimensions (see screenshot in Figure 1; the more lightweight look and feel in comparison to the mockup of D6.1 reduces complexity and highlights the actual content). These widgets help explore the story-

telling potential of big data visualization and address calls for methods to support the complementary relationship between the explorative and analytical dimensions of information visualization. The ASAP dashboard builds on insights gained from the *Media Watch on Climate Change,*[4] which initially served as a rapid prototyping platform to develop the widget synchronization, as a means to gather feedback (T6.3) from non-expert users (T6.4), and as an outreach channel (WP10). Figure 1 shows a screenshot of the actual ASAP dashboard prototype – the fully integrated version will be reported in D6.4, complemented by the usability evaluation of D6.5.

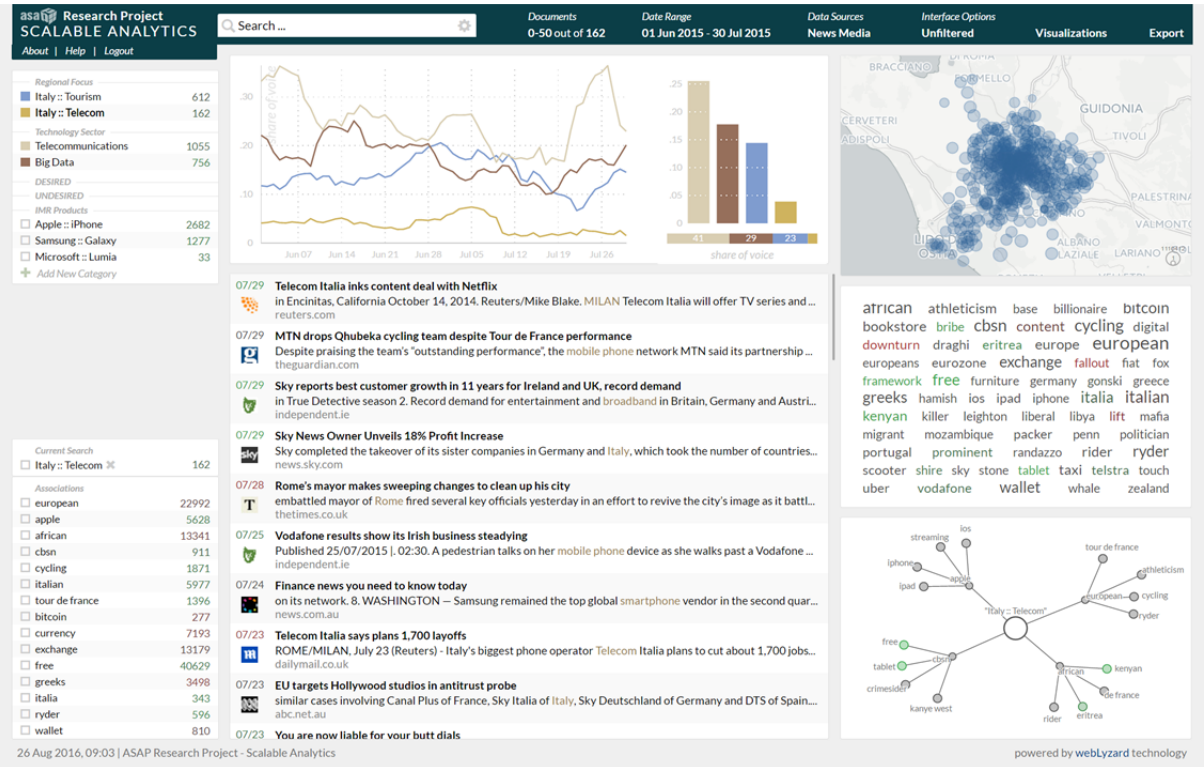


Figure 1. Screenshot of the ASAP dashboard prototype (see Section *Visual Analytics Components* for a description of the individual D6.1 and D6.2 components)

Deliverable Structure

The remainder of this deliverable summarizes the work of T6.1 – T6.4, starting with the visual analytics components, the system architecture including an outline of the developed APIs, the chosen representation of statistical data, and a summary of the visualization workflow. The visual analytics components are building blocks for full visualization dashboards and include interactive D3.js modules for line charts, donut charts, bar charts, scatter plot and geographic maps, to be synchronized within the ASAP dashboard (T6.3). This portfolio of visualizations is complemented by a slider mechanism to select time intervals for the analysis. D6.3 concludes with technical considerations in terms of rendering and indexing strategy, the scalability of the open API, and rapid deployment in distributed environments.

[4] www.ecoresearch.net/climate

Visual Analytics Components

This section summarizes the individual visualization components of WP6. While the *trend chart* and the *geographic map* build upon previous work that is being extended and refined for the purposes of ASAP, the *donut chart, bar chart* and *time interval slider* components are new developments.

D6.2 introduces a redesigned user interface with a new layout and increased feature set. The more lightweight look and feel, as compared to the initial design reported in D6.1, reduces complexity and highlights the actual content; i.e. query results and visualizations. Instead of showing all the available options at once, many features of the user interface are now hidden until being activated via hovering above the corresponding interface element. To further reduce cognitive overhead, the new navigation structure distinguishes between global, view-specific and element-specific settings and actions.

To support interactive query refinement while exploring datasets with the ASAP dashboard, adaptive tooltips and context menus have been developed in Year 2 (representing the element-specific part of the multi-layer menu structure outlined above). These tooltips offer the most relevant actions and settings in the context of the current user interaction, and always include the ability to (i) replace the search term, or to either (ii) extend or (iii) restrict the search via Boolean operators.

Trend and Donut Charts

Trend charts are central components of the webLyzard dashboard, using D3.js to render dynamic transitions – to show different source combinations, for example, or incremental changes in a dataset. Building on existing functionality from previous projects, T6.1 has rewritten and extended the trend chart module to eliminate dependencies on third-party libraries, increase rendering performance, and support a wider range of usage scenarios.

Applied to Web content repositories, the line chart shows the evolution of topics. Its vertical axis re-scales automatically. This feature is particularly useful if a high value obscures the distributions of other variables. Hovering above data points displays tooltips with context-specific metadata. For *Web content analytics* (WP8), the frequency of a topic as well as the sentiment expressed towards this topic will serve as key indicators. Derived metrics such as disagreement (= the standard deviation of sentiment) can show how contested a topic is. The term ‘tsunami’, for example, typically has a low standard deviation since everyone agrees on its negative connotation. For the WP9 use case on *telecommunications data analysis*, the trend chart will enable us to show the number of bookings or enquiries, for example, and to summarize the temporal distribution in conjunction with specific events.

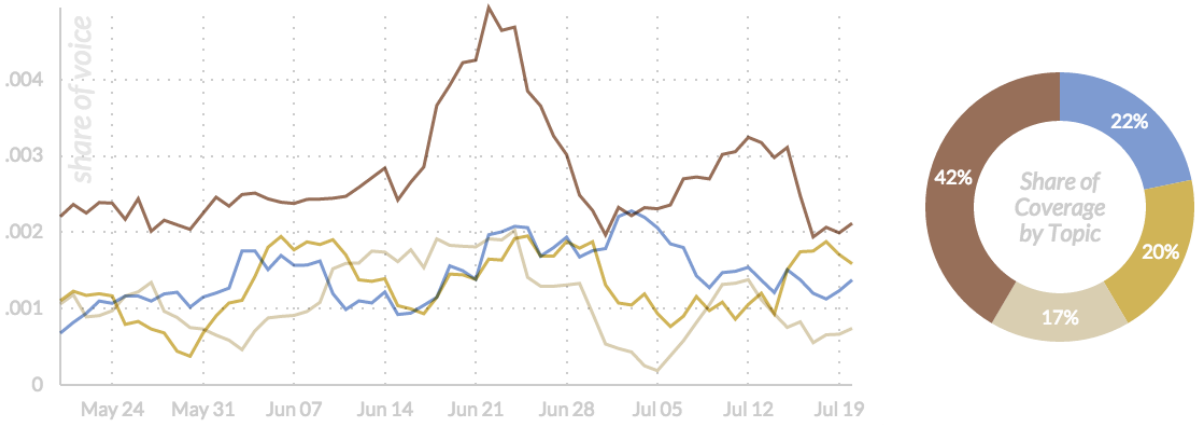


Figure 2. Trend and donut chart

The new donut chart shown in Figure 2 is functionally similar to the previously used pie chart, but improves data perception by de-emphasizing the use of areas. While the areas of the pie chart slices can be deceiving (their size grows exponential rather than a linear manner), the donut chart provides a more realistic display of attribute values. Users focus on reading arc lengths rather than comparing slice areas. An additional advantage is the use of the empty center to display additional information, for example the name of the shown variable or a chart legend.

The data export function provides time series data from the trend chart module in XLSX format or as a comma-separated text file (CSV) encoded in UTF-8 format. Specific ASAP extensions to this component include (i) the export of larger datasets with a progress bar, (ii) the ability to start several downloads in parallel, and (iii) the option to cancel an extensive download while the progress bar is being shown. This will allow the processing of large datasets in statistical packages such as R and SPSS, and a range of other external applications.

Extended Interactive Trend Chart Functionality

The *Trend Chart* is being further extended and a zooming technique is being implemented. It provides a more fine-grained display of the desired time interval, allowing an in-depth and on-the-fly analysis. Similarly to the GeoMap visualization, the Trend Chart's zooming allows the following interactions:

1. *Zoom-in/out* – triggered by a mouse wheel / trackpad device, the display is augmented/reduced by a certain factor. A double click triggers the zoom-in procedure.
2. *Side panning* – after zooming-in, user is able to navigate through the timeline via the dragging procedure.

The *Donut Chart* is planned be synced with the Trend Chart and updated according to the current selection.

Programmable Methods as Event System Hooks

As part of the full dashboard application, various visualizations, among them the trend chart and donut chart, are available for stand-alone embedding via iframes and can be configured using URL parameters. In addition to visualizing aggregation based on the document data, the trend chart has been extended to allow adding predefined statistical indicators using a format like:

`?search=rome&indicator=sms,phonecall`.

Chart legends, which were previously only available for static exports, have also been made available for the stand-alone embeddings. To support additional interaction when embedding charts in other host applications, using `postMessage()`, the iframe and the parent window can communicate with each other. The following JavaScript example demonstrates updating an embedded trend chart from the host window with a new search query:

```
iframe.contentWindow.postMessage(JSON.stringify({
  type: 'wlt.search',
  target: 'trendChart',
  searchTerm: 'solar energy',
  samples: ['news', 'social'],
  beginDate: new Date('1 Jan 2016'),
  endDate: new Date('31 Mar 2016')
}), location.href);
```

Going the other way around, a click event triggered within the embedded iframe can be emitted to the host window:

```
window.parent.postMessage(JSON.stringify({
  type: 'wlt.click',
  target: 'trendChart',
  searchTerm: searchTerm
}), location.href);
```

Bar Chart

To increase the versatility of the chart library and address the above-mentioned limitations of pie charts and donut charts, the portfolio has been extended with *two different types of bar charts* that can be deployed individually or in combination:

- *Visualizing Clustered Data:* This generic extension allows visualizing any type of grouped datasets (e.g. static information sources with a limited number of daily or weekly updates in the case of the *Web content analytics* of WP8, or different tourist segments in the telecommunications data analysis of WP9). The example shown in Figure 3 (left) compares the total number of mentions per category. Each bar is accompanied with a number that indicates the document count, and the three most common keywords.
- *Visualizing Metadata Distributions:* The inherent limitations of pie charts are only partially addressed by the above mentioned switch to donut charts. While pie charts in all their manifestations have gained a lot of popularity among end

users in software applications which offer ready-made charts, studies on a perceptual level show that bar charts are more effective for estimating differences in given values when compared to angle encodings (Heer and Bostock, 2010). Bar charts also improve the readability when rendering many different categories, and can be easily subdivided to display additional metadata such as the positive, negative and neutral sentiment shown in Figure 3 (right). The split design allows us to show both, the overall metadata distribution as well as the distribution by data category as percentages.

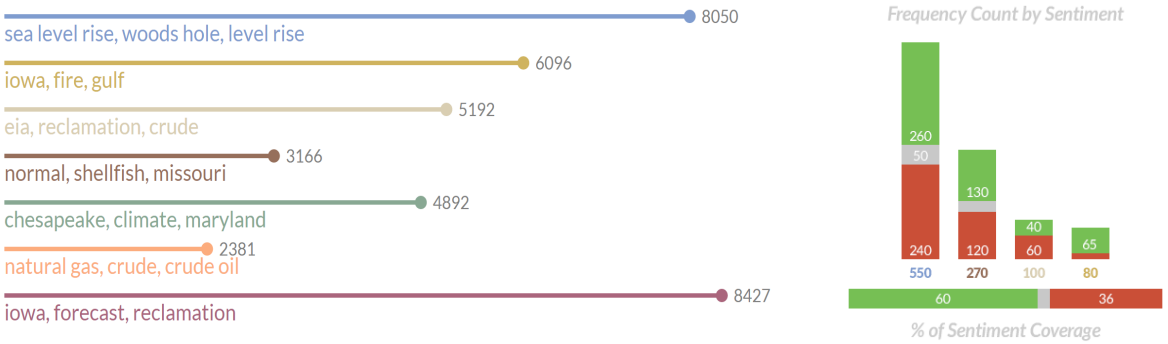


Figure 3. Bar charts present grouped datasets as an alternative to time series data (left), and as an alternative to the previous pie chart module (right)

Scatter Plot

The revised version of the webLyzard scatter plot translates two-dimensional tables into visual form, providing interactive features and animated transitions to show incremental changes from query to query. For content analytics applications, it is currently used as a "Source Map" (see Figure 4), showing the relative importance of a given topic as well as the editorial position of various authors towards this topic. In the following, we summarize the key characteristics of the scatter plot component:

- *Layout.* The sources are shown with circles of variable position, size and color. The circles are mapped on a two-dimensional layout with the horizontal axis corresponding to the frequency and the vertical axis to the sentiment value range. Both axes use a quadratic scale that distributes circles more evenly in space due to the fact that the majority of sources has small frequency and sentiment values. The size of a source is proportional to its reach value and the color represents the average sentiment value.
- *Interactive Features.* On mouse over, a tooltip provides additional information including the name of the source, numeric values of four variables (frequency, centrality, sentiment and reach), and the top 3 keywords. The view-specific menu provides an option to label the sources. The positioning of the labels is calculated with an adapted version of the D3-Labeler library (Wang, 2014), which optimizes the positioning to minimize overlap.
- *Incremental Updates.* Each query triggers an incremental update of the layout using smooth animations. The axes ticks are also calculated dynamically depending on the respective data value ranges.

Figure 4. Scatter plot showing the frequency vs. sentiment distribution of "climate change" references by international news media (2015)

Geographic Map

The geographic map shown in Figure 5 builds on previous work on the *Media Watch on Climate Change* (Scharl et al. 2013a; Scharl et al. 2013b) within the DecarboNet FP7 project,[5] which provided an initial D3.js-based implementation. Work in T6.2 extended this earlier work by providing:

- Custom base layers including raster image and vector tiles to show additional details when rendering the display. This required tools to generate raster image tiles, to serve them as MBTiles, and to customize these MBTiles according to specific use case requirements;
- adaptive tooltips and view-specific menus for data highlighting, selection, and drill down,
- a choropleth feature in conjunction with reverse geocoding on the country-level to provide additional data highlighting features, especially in regards to structured datasets,
- dynamic clustering to allow adaptive data exploration from high-level views to regional/local data,
- support to map custom metadata values to visual attributes,
- and publishing the geographic map as an open source project.

[5] www.decarbonet.eu

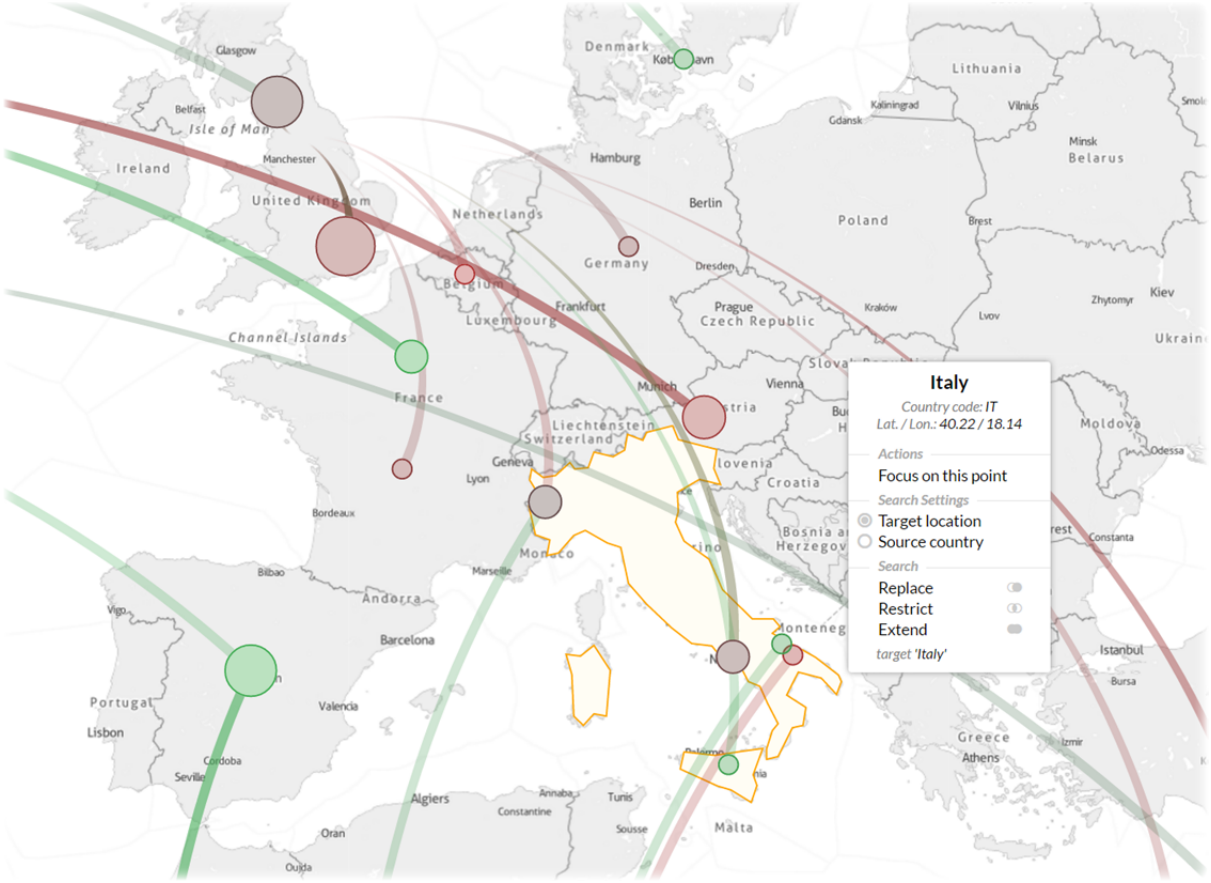


Figure 5. Screenshot of the revised geographic map module, including a custom base layer and an adaptive tooltip with user actions and Boolean query refinement options

Custom Base Layers

The pre-T6.2 version of the GeoMap module was entirely based on SVG, using country shapes as base layer to display borders and provide country-level hovering and selection. While such SVG-rendered shapes are great for a quick overview and easy to adapt in terms of visual attributes, they lack the level of detail required for the ASAP use cases – when focusing on a given limited area such as the City of Rome, for example, the shapes' lack of detail becomes obvious as county and district border shapes are notably missing.

While theoretically there are no limitations on how detailed SVG shapes can be, the actual rendering process is resource-intensive. The previously used shapes required approximately 357 kB of uncompressed data. Adding more detail to the shapes quickly grows the dataset into the range of several megabytes, which is not feasible for Web client applications – especially when taking mobile access scenarios into account. To address these constraints, T6.2 implemented a pipeline of tools to offer customized raster-based image tiles for map base layers.

- **GeoMap JavaScript Client Library.** The Web client library is extended to support both raster-based image and vector tiles as base layers. The standard base layer's subtle design was on purpose, featuring the necessary shapes

and labels to offer the geographic context, but keeping the main focus on the actual data displayed on top of the base layer. POIs can be displayed in various styles. Additionally, the rasterized base layer can be combined with SVG labels to increase the flexibility of visual applications. Finally, support for vector tiles adds additional customization options.

- **Tile Server.** The base tiles need to be hosted and served. tilestream[6] is an open source high performance map tile server based on MBTiles files. Deploying the server in combination with docker[7] and Apache HTTP Server[8] addresses scalability and security issues. docker simplifies deployment and scaling, while Apache guarantees that tile server access is limited to ASAP partners and authenticated third parties with the required permissions.
- **Customized Map Creation** ensures maximum flexibility when it comes to specific use case requirements. The ASAP map creation pipeline yields custom MBTiles files for the tile server. The format used to style maps is CartoCSS. By building upon the CSS standard, it offers a low learning curve for designers familiar with common Web technologies. These styles are used as a basis to render the tiles with mapnik,[9] and can be combined with different data sources. The default is OpenStreetMap,[10] but sources and formats can be applied based on specific requirements (e.g., ArcGIS data).

Highlighting Data Points and Connections, Selection and Drill Down

Extending the GeoMap component with adaptive tooltips and context menus enriches its functionality while ensuring a unified user experience. Based on the analyst's current context, for example in the form of a country shape or point of interest, the tooltip displays filtered information and a context menu to either drill down or extend the search. Basic highlighting functions include visual cues to show specific (groups of) data points when users hover over related components in the ASAP dashboard, or color coding of data points according to metadata attributes such as sentiment. With a special emphasis on structured datasets, the GeoMap library has also been extended with a choropleth feature as well as a reverse geocoding function that operates on the country level.

The JavaScript library of the Geographic Map in combination with tooltips and context menu as well as the updated event system allows a range of data points highlighting techniques. The user is able to gather detailed information on custom polygons (e.g. country shapes), individual data points and trajectories (by highlighting connections between source and target data points).

[6] www.github.com/mapbox/tilestream
[7] www.docker.com
[8] httpd.apache.org
[9] www.mapnik.org
[10] www.osm.org

Dynamic Clustering

The *Atlas component* was improved to support client side interactive clustering. Interactions by users with the Geographic Map's viewport trigger custom events which allow updating the level of detail within the map as well as synced components (see Figure 6). The dynamic clustering allows a drill down in real-time from high-level views to regional/local data. Additionally, an updated event system was developed to support rapid prototyping of tightly coupled views in multiple contexts. The event system is described in its own section.

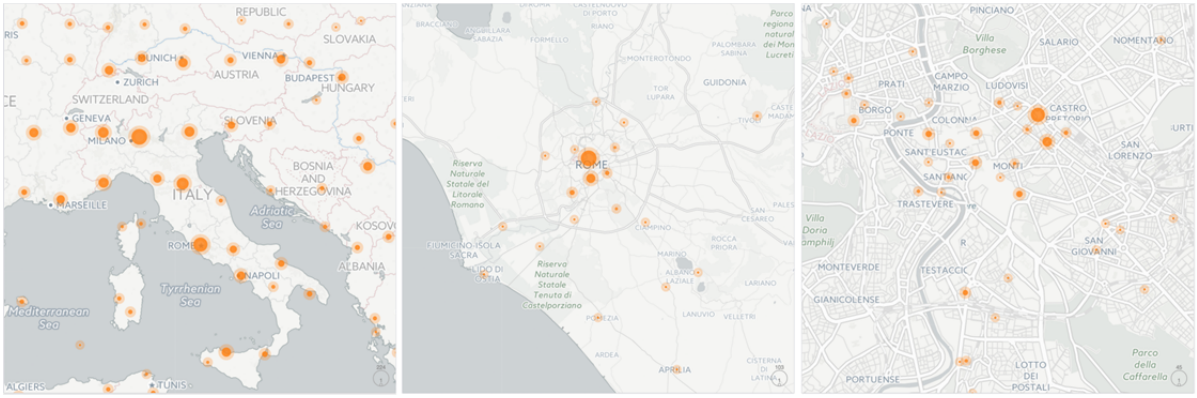


Figure 6. Example application with consecutive drill down states: The area of Rome is a single cluster in the overview viewport (left) and begins to dynamically split up into fine grained clusters while zooming in (center). Further zooming in to street level reveals more details (right). While the clustering uses a grid to define the level of detail (most obvious in the left overview viewport), individual clusters are positioned by weighting their internal location focus.

Data Overlays

Previous dashboard implementations of the Geographic Map supported the display of a very limited amount of data only, for example, the top 50 documents actually on display in the application. T6.3 introduces support for the display of multiple aggregated geographic features in the form of data overlays, immensely improving the allowed quantity of source and target locations as well as connections between them.

A stand-alone prototype including mobile cell data from WIND as well as geo-tagged Tweets has been developed to demonstrate the performance. *Circular markers* allow the comparison of multiple datasets, like phone calls and the regional distribution of social media coverage. This can be generalized to visualize arbitrary metadata for different use cases. The framework allows incremental real-time updates, triggered by user input or other external events — e.g., new content being added to the knowledge repository. This dynamic approach increases the *granularity of the display*, particularly in conjunction with improved geotagging services or data acquisition services that capture author-provided coordinates. For rapid prototyping and easy integration with other system components, the concept of *visualization containers* was implemented. The containers offer the module's full functionality including data acquisition, transformation and visualization (for follow-up exploitation efforts,

this would enable a Container-as-a-Service model in addition to the envisioned Visualization-as-a-Service approach). We describe this in more detail in the section *Containerized Visualization as a Service*.

We experimented with several ways to combine data overlay techniques and densify the information on display, keeping in mind requirements in regards to usability and data perception. The prototype shown in Figure 7 uses color to distinguish data sources as well as data point size for the data attribute. Our approach favors visual simplicity, but – as described previously – combined with the interaction and data highlighting techniques, it supports a range of advanced data exploration features. While in terms of implementation it is straightforward to combine e.g. a choropleth base layer with additional layers using circles, such combinations tend to map data to different visual attributes (e.g. country shape, circle size, color) and lead to visual clutter, therefore slowing and skewing the perception of the given results.

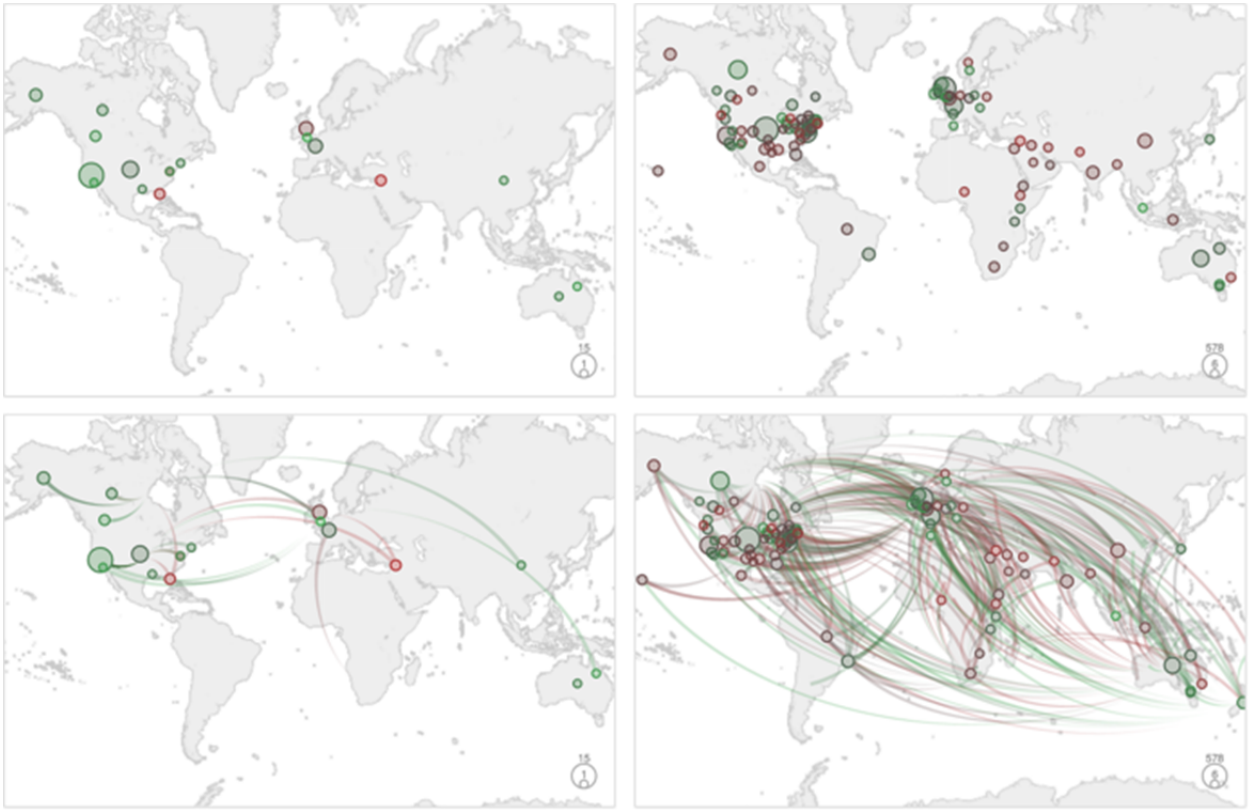


Figure 7. Comparing the results for the same data query of the previous implementation (left column) to the new approach using data aggregation and geo clustering (right column). The first row of visualizations shows data points only while the second row adds "trajectories" between source and target locations.

Custom Metadata

While the default implementation of the Geographic Map displays aggregated frequency and sentiment values, the library has been extended to support function overrides to display additional numeric metadata such as confidence values (predictive analysis, veracity detection) on visual attributes (e.g. size, color, opacity).

Open Source Library

The Geographic Map JavaScript client library is available under an *Apache 2* open source license, publicly available at www.github.com/weblyzard/infovyz. To promote adoption and ease development, the client library is accompanied by example implementations, a comprehensive documentation and programmatic test cases.

Metadata Exploration

Recursive Pattern Arrangement for Showing Temporal Views

Keim et al. (1995) propose a recursive pattern arrangement for pixel-based visualizations. One important information source for this arrangement is the calendar aspect of time, i.e. time being composed of multiple granularities such as day, weeks, or months. For pixel-based visualizations of time-oriented data, the method has become the standard arrangement. Two important shortcomings are a lack of user-orientation and the need for focus and context capabilities (Card et al,. 1999). For the latter, Shimabukuro et al. (2004) propose the multi-scale visualization. It provides focus and context, but the views are detached and the lack of orientation becomes an even bigger problem due to frequent view shifting that users need to perform. To resolve those issues, Lammarsch et al. (2009) introduced the GROOVE visualization with integrated focus and context based on granularities and the recursive pattern arrangement. D6.1 has adopted the GROOVE approach, implemented it via the D3.js library, and extended the capabilities to show additional metadata dimension such as average document sentiment.

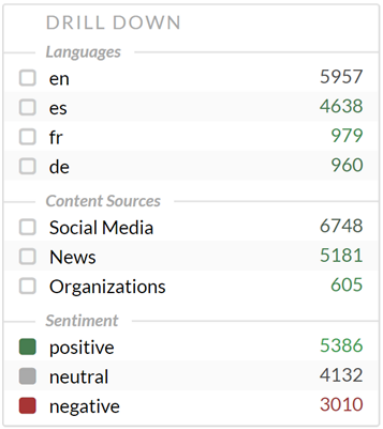
Pixel-based visualization originally used one pixel per data element (e.g., a calendar date), using color coding to represent the data value. Newer implementations such as GROOVE represent each data element by a square that might be only the size of one pixel, but automatically expands if more space is available. The pixels (or squares) are arranged according to the days of a year, similar to a calendar.

Within ASAP, the initial goal was to show both frequency and sentiment data, which exceeds the possibilities of the original GROOVE implementation. Therefore, a new color-coding scheme based on work by Choudhury et al. (2011) was developed to extend GROOVE to allow a hybrid computation of hue, chroma, and lightness.

Drill Down Sidebar

Based on initial experiments with the GROOVE visualization outlined above, which showed that most users prefer more well-known and easier to interpret interface representations, we decided to discontinue the further development of GROOVE and instead focus Year 3 efforts on a *drill down sidebar* to show temporal views and convey temporal variations in metadata attributes. The drill down sidebar has the additional advantages of providing a clear separation of attribute values – e.g., positive vs. neutral vs. negative sentiment.

To switch between different sidebar types, a new icon was added next to the header of the window. The new *Drill Down* menu consists of *"synthetic topics"* that represent metadata categories relating to the current search. It allows using the trend chart to compare elements within a metadata category (similar to selected topics as shown in Figure 8), but prohibits selections across categories to avoid confusion. Currently, the following metadata attributes are supported:



- *Language:* EN, FR, DE, ES
- *Source Type:* News Media, Social Media, etc.
- *Sentiment:* Positive, Negative, Neutral

The new navigational structure will also form the basis for the *Statistics* sidebar, in addition to the *Topics* and *Drill Down* sidebars, to be reported in D6.4: ASAP Dashboard.

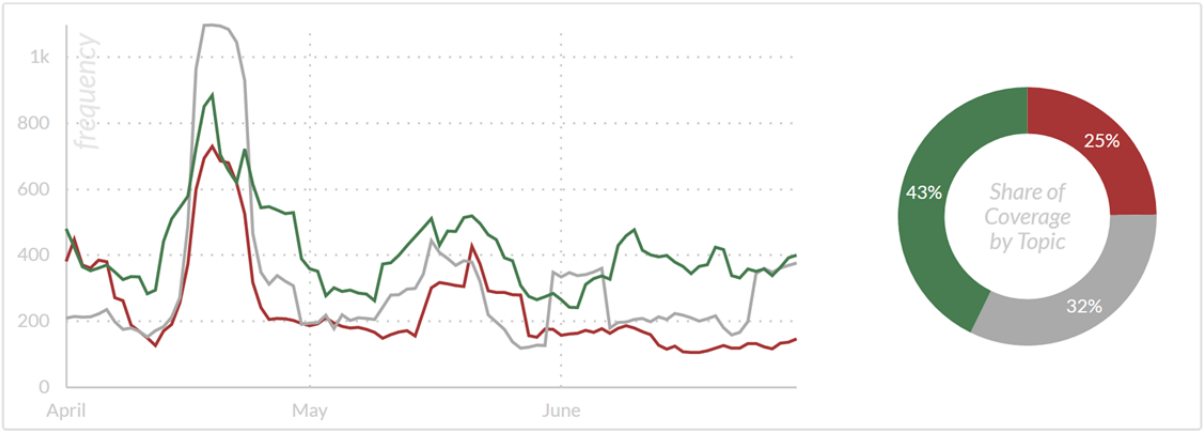


Figure 8. Metadata drill down via line and donut chart showing positive vs. neutral vs. negative sentiment for a news media query on "Rome"

Temporal Controls

Date Selection

To investigate longitudinal data, a time slider for selecting a date range for the analysis has been developed in a joint effort with researchers from the FP7 project Pheme.[11] The slider element consists of a timespan bar in conjunction with a sliding window. It displays daily amounts of documents for the full date range available in the portal. Figure 9 illustrates the timespan bar based on randomly generated data. The grid ticks on the horizontal axis provide the temporal context, with labels for years and months.

[11] www.pheme.eu

Figure 9. Time interval slider

User can modify and adjust the range in four different ways:

1) by dragging and releasing the sliding window to adjust its position;
2) by performing a click-drag-release interaction outside the sliding window to select the desired time interval;
3) by dragging the two triangular grey-colored handles situated on each side of the sliding window to resize it;
4) by clicking the two triangular arrow buttons located on both sides of the timespan bar to shift the sliding window by a fixed interval (day, week, month or year) towards the direction of the arrow. The interval will be predefined dynamically, based on the current date range: the bigger the range, the bigger the interval and vice-versa.

Any of the aforementioned interactions will set the current date range according to the new position of the sliding window and consequently update the entire portal.

Date Range Shifting

There will also exist an additional functionality to allow shifting the current date range by a fixed interval (day, week, month or year), working the same way as the two buttons on both sides of the timespan bar. However, it will be useful when the timespan bar is not visible or available. The interval will be predefined dynamically, too. This functionality will be accessible through the “Date Range” menu in the portal header.

The menu will feature two sections: “Date Picker” and “Date Range.” The first will provide two options – “Show/Hide Calendar” and “Show/Hide Timeline” – each enabling/disabling the respective component. The second will contain two options to shift the date range forwards and backwards.

System Architecture

Rendering Performance

To assess and improve the rendering and animation performance of the visualization methods, the strengths and weaknesses of different rendering techniques were investigated in Year 1 of the ASAP project. While SVG is known to perform better with fewer elements and larger rendering areas, for example, *Canvas* is superior in the case of more elements and smaller rendering areas.[12]

[12] www.smus.com/canvas-vs-svg-performance

Since rendering text in SVG is known to be prone to performance issues, we developed a simple test program to render and animate several thousand text elements, using the following techniques: D3 (SVG), D3 (Canvas), D3 (WebGL, Canvas), KineticJS (Canvas),[13] PixiJS (WebGL, Canvas).[14]

In terms of relative rendering speed we observed the best performance using PixiJS, which is a 2D rendering library utilizing the graphics card by using WebGL and rendering the output on a Canvas element. However, although slight performance improvements could be achieved using PixiJS, we assessed the various implications and decided to keep using D3 and SVG for the following reasons:

- Since SVG works with vector graphics, the output is more precise and visually more appealing compared to pixel-based graphics;
- WebGL is a fairly new technology and therefore might not be supported by all available browsers, especially on mobile devices;
- ASAP visualizations are highly interactive, where the major advantage lies with D3 and SVG, due to its data-binding and event handling capabilities;
- Straightforward implementation of animations – when the attributes of the elements to be moved are known, it is straightforward to directly select and animate them in D3. In Canvas, by contrast, all elements need to be redrawn, and one needs to keep track of the elements to be moved, and interpolate their new position (D3 manages these processes automatically).
- Text cannot be rendered directly in GL, since a texture element needs to be generated for each text element; this complicates animating font size changes as compared to SVG (with respect to text quality and correct positioning);

While SVG offers significant advantages for web-based applications, there are specific scenarios where it lacks performance in comparison to WebGL implementations – if a lot of textual elements are required, for example, it loses its performance advantage and has issues with rendering quality (see Section “Rendering Process” below). We investigated possible WebGL implementations, as the geographic features of the D3.js library itself can be used to create shapes, but the data can also be passed on to a WebGL renderer such as three.js instead of SVG. As an alternative, NASA World Wind (formerly a Java-based application framework only) is currently being ported to HTML5 and JavaScript.[15]

In light of the potential drawbacks outlined above, we considered the gains in raw performance not significant enough to justify the efforts required to further test and potentially adopt Canvas/WebGL-based approaches.

[13] www.kineticjs.com
[14] www.pixijs.com
[15] www.webworldwind.org

Indexing Strategy and Deployment

The visualization components of ASAP are based on high-performance queries on unstructured and structured data repositories. Before the start of the ASAP project, the webLyzard processing pipeline was based on Apache Lucene[16] and largely built around the processing of large datasets in batch mode. We increased flexibility through a modularization strategy, together with a migration to Elasticsearch,[17] a distributed search and analytics engine made available under an Apache 2 open source license. Elasticsearch provides a RESTful API using JSON over HTTP. Built for the cloud, it ensures the required scalability for real-time queries that provide the data for through multi-tenancy and sharded indexing.

Elasticsearch not only speeds up accessing domain-specific repositories of unstructured content, but also facilitates the process of slicing statistical linked data and their integration with other data sources. Elasticsearch aggregations yield additional information for user queries or API calls:

- ***top k answers****.* Top search results based on a simple or advanced query;
- ***data slices****.* Most often across time, but in the case of complex datasets other dimensions can be included as well – when analyzing tourism transactions, for example, one can obtain all the flights originating from a specific airport to multiple destinations;
- ***data summaries****.* Indicator performance is highlighted not just by color coding, but also through a data summary.

On top of Elasticsearch, the API specified in Annex C of this deliverable is being implemented using Vert.x[18], a high-performance, lightweight and scalable application framework. Using its distributed event bus, larger applications can be broken down into micro-services, which can then be horizontally scaled over multiple hosts. As a non-blocking, asynchronous toolkit, Vert.x can easily scale to several thousand requests per second. The modular processing strategy of WP6, in conjunction with the provision of Vert.x-based RESTful APIs, will enable the on-the-fly integration of intermediate results.

For a more flexible deployment of ASAP visualization components, Kernel-based Virtual Machines (KVMs)[19] were replaced with Docker instances (see Figure 10).[20] Since the Docker engine container comprises just the application and its dependencies (in contrast to KVM, which also includes the guest operating system), this will increase the portability and efficiency of the developed components. The migration to a distributed Docker architecture has been completed in the third quarter of 2015.

[16] lucene.apache.org
[17] elastic.co
[18] vertx.io
[19] www.linux-kvm.org
[20] www.docker.com

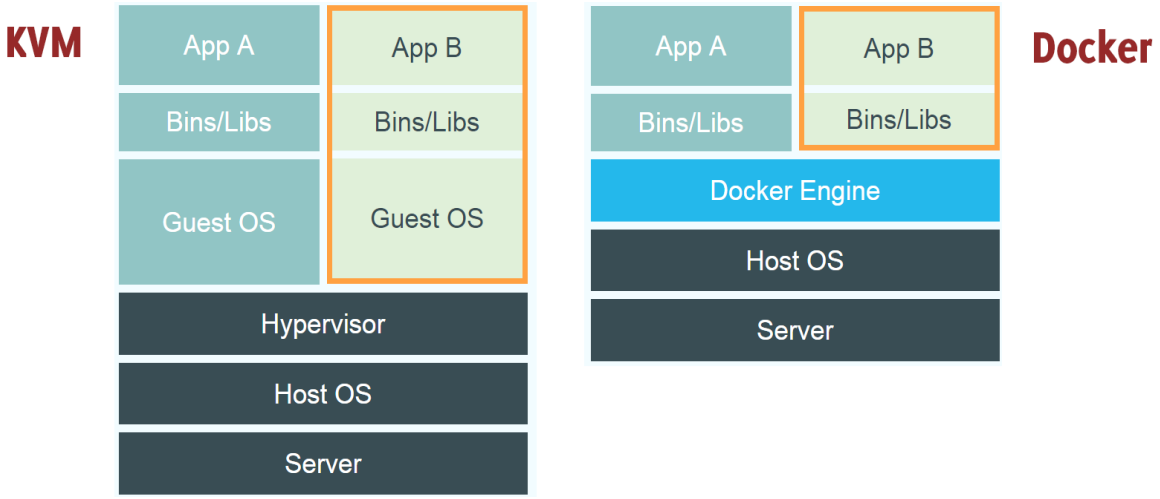


Figure 10. Virtual machines versus Docker containers (Source: www.docker.com)

Application Programming Interface

The ASAP visualization pipeline for processing unstructured and structured content from multiple sources aims to solve the data matching problem. It is being made available via an open API for members of the consortium, and selected external partners who will help evaluate the provided functionality. To align the APIs with the specific requirements of the consortium partners, a stand-alone *Request for Comments* was sent out in July 2015. An extended version was then provided as part of the initial draft of D6.2 in August 2015. Future versions of the API will evolve alongside the use case requirements, and play an important part in formulating the ASAP exploitation strategy (T10.2), which will leverage the functionality of the API pursuing a Visualization-as-a-Service (VaaS) approach.

Advantages. Prior to ASAP, the webLyzard dashboard existed in different versions, fully customized to client specifications and typically including a multiple visual tools. The manual effort of customizing each system quickly became a cost driver that does not scale well with an increasing number of projects. When aiming to support flexible on-the-fly visualizations based on dynamic datasets, as required by ASAP, this approach is no longer feasible at all. This led to the development of several open APIs, bundled within a uniform framework for the rapid integration of heterogeneous data sources into a common visualization processing pipeline.

Specification. The specification of the APIs aimed to provide a simple and unified interface through which to expose all data and interface services, increasing the degree of automation between various components and offering a clean and extensible set of JSON formats for each important service. This enables rapid deployment on a wide range of platforms, and provides instant access to relevant data – not only to serve as a building block of the use case applications, but also to support the ongoing development process.

For developers, the APIs structure and streamline the overall workflow. A visualization designer does not need to learn about SPARQL queries to retrieve and visualize entities referenced in a text. Similarly, a backend developer does not need to know about the specifics of a certain visualization to demonstrate an improved knowledge extraction method.

The main API convention is that content to be ingested by the visualization engine can be modelled as a small set of objects (*visualizations*, *queries*, *documents*, *annotations*, *statistical observations)* in JSON format. Each API is designed to be flexible and future-proof, following a versioning policy and requiring users to be authenticated. The APIs are in an early stage; future versions will not fundamentally change their structure, but will certainly refine them and add specific functions. The current set of APIs reported in Annex C of D6.2 comprises the following:

- *Document API* – ingests unstructured data, for example crawled Web documents from WP8. The main objects are *Documents*, *Sentences* and *Annotations*. This API can be used for sharing documents regardless of their provenance, as well as annotations from knowledge extraction services (sentiment analysis, named entity detection, etc.).
- *Statistical Data API* – ingests structured data from a variety of sources, for example the telecommunications data of WP9. The main object is a *Statistical Observation*. The API follows the RDF Data Cube philosophy, splitting statistical data into datasets with the same structure and components (dimensions, measures, attributes, observations).
- *Search API* – returns a set of query results in the form of unstructured text documents. The main object is *Query*. The next version of the API to be reported in D6.3 will also support queries for structured datasets.
- *Embeddable Visualization API* – provides the ASAP partners with a means to integrate visualizations in their applications, typically based on the results of a search query. The main object is *Visualization*.

Document API (Unstructured Data)

The *Document API* ingests different types of textual content (binary, text, html, etc.). It does not require fully annotated texts. *Sentences* (POS lists, token lists, etc.), *annotations* (sentiment, named entities, etc.) or *metadata* (data about the file itself, e.g. provenance) can be added in advance by a partner, or they can later be provided in a separate process via the *Annotation API*.

The creation of independent Elasticsearch[21] repositories (see Section on “Technical Considerations” below) allows distinguishing datasets from different tasks or partner organizations. Each repository has a unique ID. The Document API is used for up-

[21] www.elastic.co

loading documents to such a repository. When adding new documents, one will have to set the content, content type, provenance, time, language and location of the document, along with an optional set of annotations. Annotations can later be added either via webLyzard's own knowledge extraction components (Scharl et al. 2016) including sentiment analysis (Weichselbraun et al., 2013, 2014) and named entity recognition (Weichselbraun et al., 2015), using the text mining operators of IMR's *Mignify* platform (see Section 2.2 of D8.2), or by using third-party services.

Statistical Data API (Structured Data)

The analytical goals of ASAP and the requirement of visualizing structured data from multiple sources (e.g. time series of Web content metrics from WP8 in conjunction with telecommunications data from WP9) required a new *data format* that supports:

- datasets created by ASAP partners, using common formats such as CSV, JSON, RDF, XML, etc.;
- complementary data from Open Data Initiatives (e.g. governmental open data, or indicators from international organizations);
- RDF Data Cubes in the SDMX, QB and other similar RDF formats.

This format should resemble *the* JSON format used by the *Document API* to ensure consistency across structured and unstructured data, and the ability to visualize these data along multiple dimensions. To represent statistical data from a variety of sources, we adopted the RDF Data Cube Vocabulary (QB) approach, which is the current standard for publishing statistical data in RDF format (see Appendix B). By splitting the statistical data into cubes with a maximum of three dimensions (e.g., mobile cellular subscriptions, location and time), QB offers a simple structure that can be shared by all datasets. In order to use data from multiple datasets, there are several possible strategies:

- determine whether all the components of the respective datasets match using complex *ontology alignment* or data matching techniques;
- use *machine learning techniques* to perform the data matching automatically;
- create a *common data format* for all datasets based on the basic concepts of this vocabulary – i.e., organized in datasets which contain observations described through measures, dimension, attributes, etc.

When developing the API for ASAP, we have chosen the third option (follow-up projects might add machine learning techniques to perform complex data matching, for example to ingest *noisy datasets* with uncertain provenance). The RDF Data Cube philosophy is not only useful for QB or SDMX datasets, but also for integrating *any type of statistical data*. The underlying idea was to use a common statistical mapping with a rich set of optional fields to visualize various ASAP datasets using a common Statistical API. This mapping process is described in the following table.

Table 1. Mapping between QB data and Elasticsearch indices

RDF Data Cube Vocabulary Concept	Elasticsearch Correspondence
Dataset	Repository
Data Structure Document	Mapping
Code Lists	There is no need to keep code lists, but they can be converted to corresponding data types.
Statistical Observation	Document with type = “observation”
Measure Components, Dimension Components, Attributes	Fields with the corresponding Elasticsearch data types
Slices	Queries

While it appears that measure and dimension components are meshed together, this hardly happens in reality, as the typical queries would almost always involve just the dimension components (Query 1), but it does offer the possibility of querying the other components as well (Query 2):

- *Query 1 – Retrieve the growth of mobile phone users [fixed dimension] in Italy [fixed dimension] between 2005 and 2015 [time interval; free dimension].*
- *Query 2 – Retrieve the units of measurements used in this dataset.*

When uploading data with the Statistical Data API, the relevant data snippets have to be transformed into observations, which themselves are grouped into datasets. In general it is recommended to upload entire datasets instead of individual observations, but modifying individual entries is possible. This conversion process becomes particularly valuable when slicing and grouping observations via the Elasticsearch Query DSL (Domain-Specific Language), as the performance of this DSL has a significant impact on overall *scalability*.

The main advantages of using the RDF Data Cube standard for modelling statistical datasets to be ingested into the ASAP visualization processing pipeline are:

- using a proven methodology (W3C Recommendation) to drive both the data modelling and the visualization process;
- the ability to perform most operations (slice, dice, aggregations, subtotals, etc.) one would be able to perform on data cubes, which are already well-understood and formalized through *cube algebras* like the ones presented by Gray et al. (1997) and Cifferi et al. (2013);
- effective integration by aligning RDF Data Cube concepts and Elasticsearch core concepts.

Search API

The *Search API* enables data analysts to use powerful queries based on the full text of a document, its metadata attributes, and statistical observations – therefore it is the ideal means for providing data for *embeddable visualizations (see below)*.

It is a stateless, RESTful API, accessed via HTTPS that uses JSON as input and output format and supports most functions of Elasticsearch (in addition to specific operations related to the available metadata).

The Search API supports a wide range of queries such as *bool, phrase, regexp, term, wildcard, range, sentiment, date*, etc. The current version returns search results as documents. Future version might also support more granular queries for sentences, sentence combinations or paragraphs.

Embeddable Visualization API

The *Search API* allows users to access relevant data, and embed results in third-party applications. To identify hidden patterns and to better understand the underlying processes, however, visual methods are often superior.

The Embeddable Visualization API complements the rich and more complex functionality offered by the multiple coordinated views of the ASAP dashboard. It supports the integration of individual visualizations into use case applications, and represents the basic building block of ASAP's Visualization-as-a-Service (VaaS) exploitation strategy (WP10).

Calls to this API allow specifying the *appearance* (width, height, style, etc.), *data format* (html, csv, json, xml, etc.), *query* or *type of chart* that should be displayed. It works with both structured and unstructured data, and the queries resemble those sent to the Search API (see above).

Details about the new and revised visual tools to be made available both via the ASAP Dashboard as well as the Embeddable Visualization API will be described in the next section of this deliverable.

Visualization-as-a-Service (VaaS) Architecture

To support the usage of the visual analytics components in multiple contexts, a *event and notification system* was developed. In contrast to the legacy implementation used in previous versions for example of the dashboard and geographic map component, the new system doesn't require hard event dependencies inside single modules. This allows in combination with the concept of *Containerized Visualization as a Service* the rapid development of customized visualization dashboards.

Event and Notification System for Client-Side Micro Services

The visualization dashboard application includes an event system which allows communication between separate modules. The technology allows the creation of multiple coordinated views (Hubmann-Haidvogel et al., 2009). In the past, this approach worked well for the main dashboard application. However, updated system requirements (e.g. embeddable visualizations) and progress in web technologies (e.g. the rise of mobile applications) surfaced some shortcoming in the previous implementation, which uses an event system to manage communication between modules by using a register/notify pattern. Modules could subscribe to certain events and trigger an action should the event be called using the notify() function in another module. The resulting graph of dependencies among modules is shown in Figure 11.

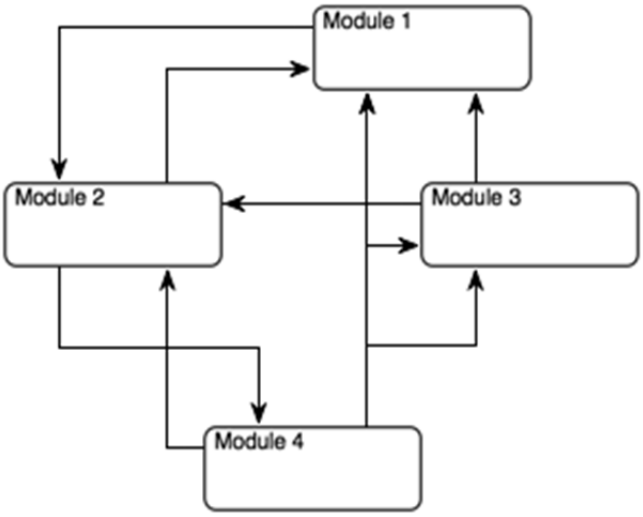


Figure 11. Exemplary dependency graph of modules following the register/notify pattern

While this approach proved to be flexible, there were some shortcomings which needed to be addressed to support future use cases:

- The event subscriptions and triggers were defined within modules which lead to hard-coded dependencies. This prohibits the use of modules in multiple contexts or applications. So because the events were hard-coded for the desktop dashboard application, there was no way to use the same event system for example for an alternate mobile application.
- While it was simple to add multiple subscriptions to one event trigger in multiple event receiving modules, it had a negative impact on the ease of code maintenance. Because each event subscription was handled within each module, there was no simple way to get an overview to a single event's subscriptions.

To address the described shortcomings, a successor event framework was developed. It had to fulfill the following requirements:

- Provide support for intra-module communication in multiple contexts (e.g. desktop and mobile application, embeddable visualizations, modules acting as plugins for other host frameworks),
- allow developers to easily get an overview of the event flow, meaning to see at a glance both an event subscription and all actions triggered by the event
- and offer support for event bubbling in nested module hierarchies.

The resulting framework consists of a combination of code building blocks and best practice patterns:

- The event flow allows modules to be loosely coupled, avoiding hard-coded dependencies between modules.
- A module should not contain logic calling other modules in the context of event triggers (this does not apply to nested or inherited modules).
- Event names should be about the actual action/event being triggered inside the module (e.g. “termStateChange”) and should avoid referencing other modules (like “notifyGeographicMapModule”).
- Modules expose event listeners (instead of the previous notify()) and public methods (instead of register()), as shown in Figure 12.
- The actual implementation of event subscriptions and action triggers (module method calls) happens outside a module’s scope, therefore allowing the use of a module in multiple contexts. Depending on the complexity of an application, optional multiple event controllers can be used to organize the event flow (Figure 13).

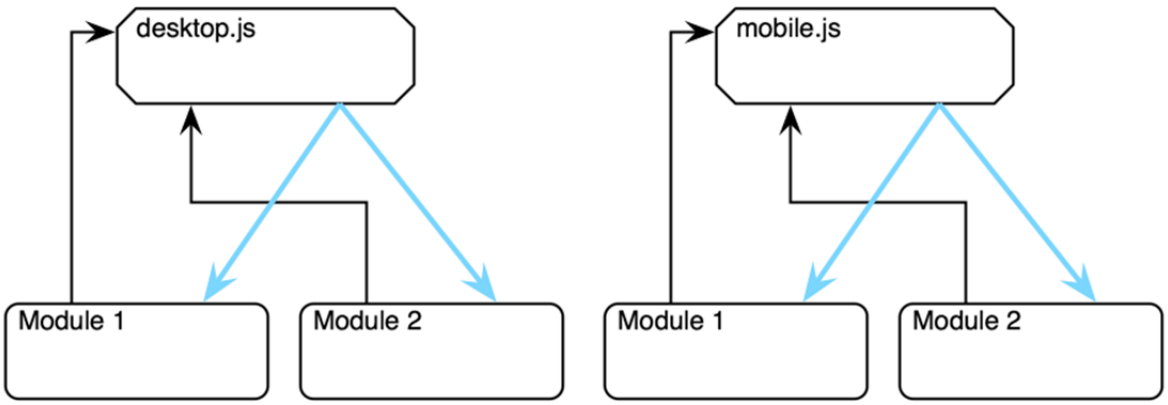


Figure 12. The diagram shows a simple event flow pattern between two loosely coupled modules used in two different applications. Event listeners (black arrows) can be subscribed to by a simple host application, for example a desktop or mobile version of a web application. Function callbacks inside event listeners can then trigger public module methods (blue arrows). Because there are no hard-coded dependencies between the modules, the same event listeners can be used multiple times in different contexts.

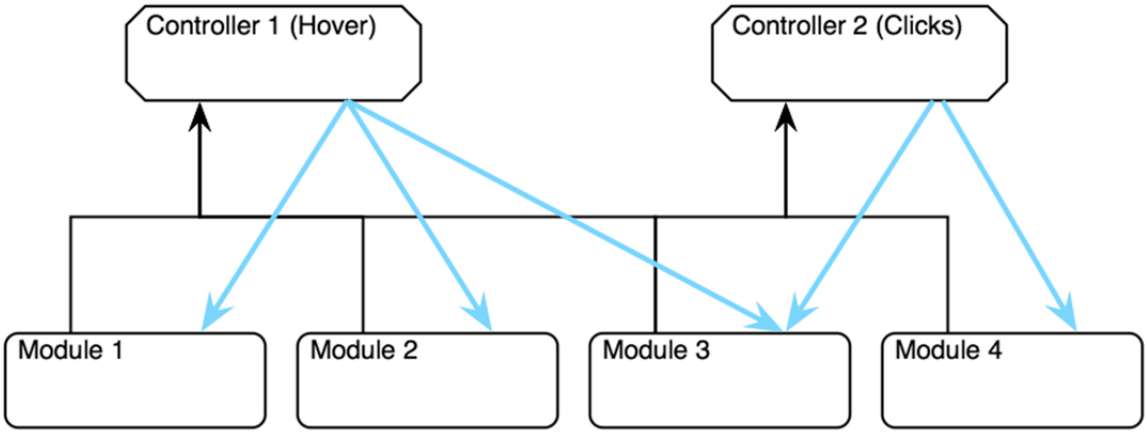


Figure 13. A more complex setup with additional modules and multiple controllers is shown in this diagram. The same event features can be applied in this application. Multiple controllers allow the separation of distinct event patterns within a larger application, therefore simplifying code management and maintenance.

The updated event system was used in combination with the already described visual analytics components to develop the required visualization interface for WP9, the *Telecommunication Data Analytics* (TDA) application use case developed by WIND. It allows the exploration of a multi-layer data set consisting of cell phone data as well as geo located tweets, as shown in Figure 14.

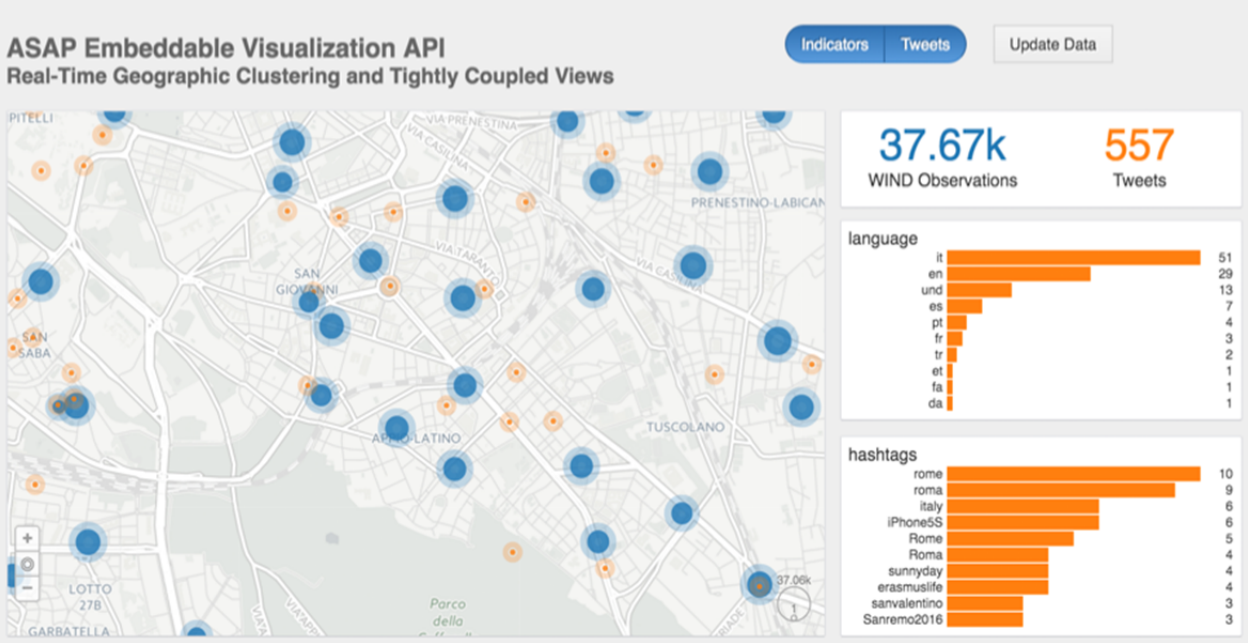


Figure 14. Screenshots of the visualization component for the WP9 Telecommunication Data Analytics (TDA) application. It demonstrates the usage of the updated event system to sync individual dashboard components in real-time based on user interactions. When the user interacts with the geographic viewport both the clustered data points in the viewport as well as the corresponding information in the other elements updates in real-time.

Containerized Visualization as a Service

Most open source charting and visualization libraries are only concerned with client side representation of data in the browser. Additionally, some libraries lack capabilities to fully support an extended data lifecycle. They are able to render a given dataset but fail when trying to properly update a given view or render data streams. While we initially decided to use the NVD3 library as a basis to develop our visualizations, it emerged during development that it lacked both described shortcomings. NVD3's underlying visualization framework d3 has the capabilities though to support updating existing charts properly when its enter/update/exit pattern is implemented correctly. Since we gained significant expertise of d3 while implementing custom visualizations, we decided to implement our own versions of standard charts (line chart, donut chart, bar chart) to properly support dynamic updates.

Additionally we came up with a lightweight framework to support a more holistic data lifecycle including data acquisition, storage and representation. The goal was to create both a setup which is easily understood and maintainable by visualization engineers as well as deployable as a building block in service driven architectures. Figure 15 shows an overview of the setup.

The main wrapper is a Docker Compose setup, which offers additional features to create setups consisting of multiple interconnected containers which can be run with one simple startup command.[22] The individual containers solve the requirements of data acquisition, storage and representation:

- **Acquisition:** A node.js based REST API server offers an interface to inject and read data into and from the storage. Additionally, being based on JavaScript itself, it allows frontend developers to implement necessary data transformations.
- **Storage:** An Elasticsearch container is both responsible for data storage and analytics. While one of Elasticsearch's core purposes is search applications, its aggregation framework was designed to offer powerful analytics features.
- **Representation:** This container is based on the nginx httpd server. It hosts static files and acts as a proxy to the REST API container.

This setup is suitable to fulfill multiple needs:

- **Rapid Prototyping:** With the stack being entirely based on JSON and JavaScript, this setup offers visualization engineers a complete environment to rapidly develop visualization prototypes extending the pure representation layer which most data visualization libraries offer only. By using Docker Compose, developers are able to run to the whole setup with a single command on a variety of development environments.

[22] www.docker.com/what-docker

- **Simple Deployment:** One of Docker's advantages is a reproducible application environment for both the developer and the production server. Multiple visualization container combinations can be deployed to a single production host system, without compromising, for example, the configuration of the hosts or other container httpd servers.

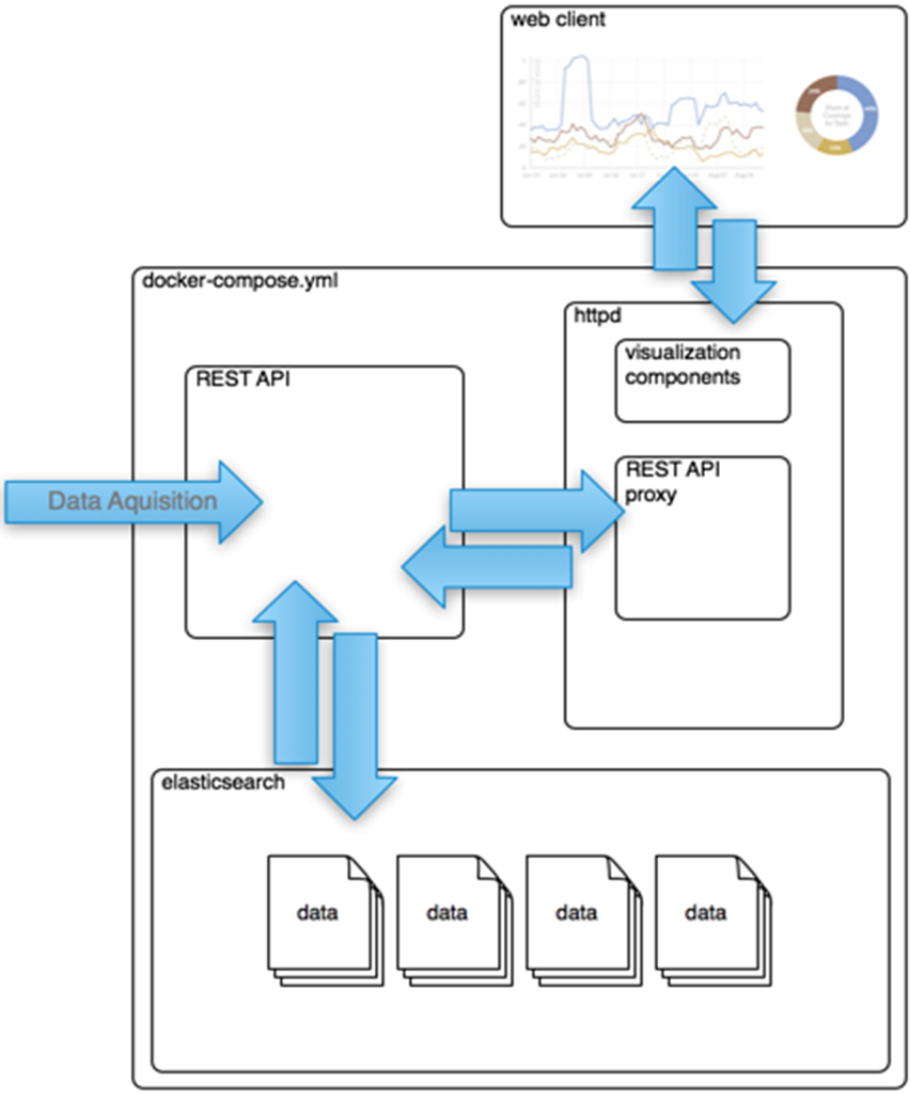


Figure 15. Overview diagram of containerized visualization with support for data acquisition, transformation and visualization.

Outlook and Next Steps

This deliverable summarizes the work conducted in WP6 of the ASAP FP7 project. In the first 30 months, this work has focused on methods to collect, represent and visualize structured as well as unstructured data, to be linked together via the ASAP dashboard. This included the development of an open API and a corresponding indexer, as well as visual analytics components for the rapid rendering of complex datasets using visual cues (e.g., color coding) to show metadata attributes, and interac-

tive mechanisms to select appropriate timescales. These components include (i) a charting module with adaptive tooltips and context menus that offer actions and settings based on the sequence of user actions, providing a user-friendly way to trigger on-the-fly query refinements; and (ii) a geographic map, where T6.2 added custom base layers by generating raster image tiles, serving them as MBTiles, and customizing the MBTiles according to the use case requirements. To render statistical data on top of the geographic map, T6.2 developed two distinct features: (ii.a) a choropleth display in conjunction with reverse geocoding on the country level to provide additional data highlighting features, especially in regards to structured datasets; (ii.b) circular markers to compare multiple datasets, e.g., phone calls and the regional distribution of social media coverage. Both features allow incremental real-time updates, triggered by user input or other external events — e.g., new content being added to the knowledge repository. For rapid prototyping and easy integration with other system components, the concept of visualization containers was implemented. The containers offer the module's full functionality including data acquisition, transformation and visualization (for follow-up exploitation efforts, this would enable a *Container-as-a-Service* model in addition to the previously envisioned *Visualization-as-a-Service* approach). Additionally, the geographic map supports vector tiles and advanced real-time features to compare multiple datasets. Finally, the geographic map was released as an open source project.

The open API allows integrating the WP6 visualization engine with the components developed in other WPs, and rendering multi-source data for the use case applications. To set the stage for a seamless integration, Appendix C of this deliverable contains the updated specification of the API, building upon the Year 1 work on processing statistical data. It allows connecting the data streams to either individual visualizations, or to the entire ASAP dashboard.

Regarding T6.3, a first fully functional prototype of the ASAP dashboard has been deployed. Adaptive tooltips and view-specific context menus were added to support on-the-fly query refinements, and to provide various options for data highlighting, selection, and drill down. A REST Application Programming Interface (API) integrates the visualization engine of the dashboard with other WPs. First published as part of D6.2 and continuously refined since then in terms of functionality and scalability, it can be subdivided into the Document API, the Annotation API, the Search API, the Statistical Data API, and the Visualization API. This allows ingesting product offers from WP8 and call data records from WP9, for example, and embedding visualizations in various analytic applications. The Elasticsearch cluster powering both the dashboard itself as well as the new APIs has been optimized for improved performance and resilience: (i) reduced memory load of individual nodes through data structure changes within the engine and the dashboards document mappings; (ii) automated query optimizations for resource-heavy aggregations; (iii) monthly index slices to reduce the size of active indices to answer queries, and improve horizontal scaling across multiple hosts for parallel queries.

We will also continue to evolve the API specification itself, as it will play an important role in the ASAP exploitation strategy (T10.2) by supporting a flexible and scalable Visualization-as-a-Service (VaaS) approach. Containerized visualizations enable the integration of WP6 components into the data analytics workflow of T5.3 and the Telecommunication Data Analytics (TDA) application of T9.3. The VaaS event and notification system offers a framework to support the manipulation of intermediary and final data sets.

References

Bostock, M., Ogievetsky, V. and Heer, J. (2011). “D3: Data-Driven Documents”, IEEE Transactions on Visualization and Computer Graphics, 17(12): 2301-2309.

P. Bonacich (2007). “Some unique properties of eigenvector centrality”, Social Networks 29(4): 555-564, 2007.

Brasoveanu, A.M.P., Sabou, M., Scharl, A., Hubmann-Haidvogel, A., Fischl, D. (2017). “Visualizing Statistical Linked Knowledge for Decision Support”. Semantic Web Journal, IOS Press, 8(1).

Brunetti, J. M., Auer, S., García, R., Klímek, J., & Nečaský, M. (2013). “Formal linked data visualization model”. In Proceedings of International Conference on Information Integration and Web-based Applications & Services, ACM, USA: 309-324.

Buitelaar, P., Cimiano, P., Frank, A., Hartung, M., & Racioppa, S. (2008). “Ontology-based information extraction and integration from heterogeneous data sources”. International Journal of Human-Computer Studies, 66(11), 759-788.

Cammarano, M., Dong, X., Chan, B., Klingner, J., Talbot, J., Halevy, A., & Hanrahan, P. (2007). “Visualization of heterogeneous data”. Visualization and Computer Graphics, IEEE Transactions on, 13(6), 1200-1207.

Card, S.K., Mackinlay, J.D. and Shneiderman, B. (1999). Readings in Information Visualization: Using Vision to Think. San Francisco: Morgan Kaufmann.

Choudhury, A., Potter, K., Rhyne, T., Livnat, Y., Johnson, C., and Alter, O. (2011). “Visualizing Global Correlation in Large-Scale Molecular Biological Data”, 1st IEEE Symposium on Biological Data Visualization (BioVis-2011). Providence, USA.

Ciferri, C., Ciferri, R., Gómez, L., Schneider, M., Vaisman, A., & Zimányi, E. (2013). “Cube algebra: a generic user-centric model and query language for OLAP cubes”. International Journal of Data Warehousing and Mining (IJDWM),9(2), 39-65.

Demiralp, C., Scheidegger, C. E., Kindlmann, G. L., Laidlaw, D. H., & Heer, J. (2014). “Visual embedding: A model for visualization”. Computer Graphics and Applications, IEEE, 34(1), 10-15.

Giese, M., Calvanese, D., Haase, P., Horrocks, I., Ioannidis, Y., Kllapi, H., Koubarakis, M., Lenzerini, M., Moller, R., Ozcep, O., Muro, M.R., Rosati, R., Schlatte, R., Schmidt, M., Soylu, A., and Waaler, A. (2013). “Scalable end-user access to big data”. Big Data Computing, 205-245.

Gray, J., Chaudhuri, S., Bosworth, A., Layman, A., Reichart, D., Venkatrao, M., Pellow, F. and Pirahesh, H. (1997). “Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-totals”. Data Mining and Knowledge Discovery, 1(1), 29-53.

Guo, P. J., Kandel, S., Hellerstein, J. M., & Heer, J. (2011). “Proactive wrangling: Mixed-initiative end-user programming of data transformation scripts”. 24th annual ACM symposium on User interface software and technology. ACM, USA: 65-74.

Heer, J. and Bostock, M. (2010). Crowdsourcing Graphical Perception: Using Mechanical Turk to Assess Visualization Design. 28th ACM Conference on Human Factors in Computing Systems (CHI-2010). Atlanta, USA: 203-212.

Heer, J. and Shneiderman, B. (2012). “Interactive Dynamics for Visual Analysis”, Communications of the ACM, 55(4): 45-54.

Hubmann-Haidvogel, A., Scharl, A. and Weichselbraun, A. (2009). “Multiple Coordinated Views for Searching and Navigating Web Content Repositories”, Information Sciences, 179(12): 1813-1821.

Kämpgen, B., & Harth, A. (2014). “OLAP4LD–A Framework for Building Analysis Applications Over Governmental Statistics”. In The Semantic Web: ESWC 2014 Satellite Events. Springer International Publishing, Berlin: 389-394.

Keim, D., Ankerst, M., and Kriegel, H.-P. (1995). "Recursive pattern: A technique for visualizing very large amounts of data", 6th Conference on Visualization (Vis-95). Atlanta, USA: 279-287.

Kelleher, C. (2014). Visualizing the universal data cube. Doctoral dissertation, University of Massachusetts Lowell.

Koudas, N., Sarawagi, S., and Srivastava, D (2006). “Record linkage: similarity measures and algorithms”. In Proceedings of the 2006 ACM SIGMOD international conference on Management of data, ACM, pp. 802–803.

Lammarsch, T., Aigner, W., Bertone, A., Mayr, E., Gartner, J., Smuc, M. and Miksch, S. (2009). “Hierarchical Temporal Patterns and Interactive Aggregated Views for Pixel-based Visualizations”. 13th International Conference Information Visualisation. Barcelona, Spain: 44-50.

Scharl, A., Hubmann-Haidvogel, A., Weichselbraun, A., Lang, H.-P. and Sabou, M. (2013a). Media Watch on Climate Change – Visual Analytics for Aggregating and Managing Environmental Knowledge from Online Sources. 46th Hawaii International Conference on Systems Sciences (HICSS-46). Maui, USA: IEEE Press: 955-964.

Scharl, A., Herring, D., Rafelsberger, W., Hubmann-Haidvogel, A., Kamolov, R., Fischl, D., Föls, M. and Weichselbraun, A. (2016). “Semantic Systems and Visual Tools to Support Environmental Communication”, IEEE Systems Journal: Forthcoming (Accepted 31 July 2015).

Scharl, A., Hubmann-Haidvogel, A., Sabou, M., Weichselbraun, A. and Lang, H.-P. (2013b). “From Web Intelligence to Knowledge Co-Creation – A Platform to Analyze and Support Stakeholder Communication”, IEEE Internet Computing, 17(5): 21-29.

Scharl, A., Weichselbraun, A., Göbel, M., Rafelsberger, W. and Kamolov, R. (2016). "Scalable Knowledge Extraction and Visualization for Web Intelligence", 49th Hawaii International Conference on System Sciences (HICSS-2016). Kauai, USA. Forthcoming (Accepted 17 Aug 2015).

Shimabukuro, M., Flores, E., de Oliveira, M., and Levkowitz, H. (2004). “Coordinated Views to Assist Exploration of Spatio-Temporal Data: A Case Study”. 2nd International Conference on Coordinated and Multiple Views in Exploratory Visualization, São Paulo, Brazil: 107-117.

Shneiderman, B. (1996). “The Eyes Have It: A Task by Data Type Taxonomy for Information Visualizations”. IEEE Symposium on Visual Languages. Boulder, USA: IEEE Press: 336-343.

Stonebraker, M., Bruckner, D., Ilyas, I. F., Beskales, G., Cherniack, M., Zdonik, S. B., and Xu, S. (2013). “Data Curation at Scale: The Data Tamer System”. In CIDR, Asilomar, California, USA, 2013.

Trinh, T. D., Do, B. L., Wetz, P., Anjomshoaa, A., & Tjoa, A. M. (2013). “Linked widgets: an approach to exploit open government data”. International Conference on Information Integration and Web-based Applications & Services. ACM, USA: 438.

Wang, E. (2014). “A D3 Plug-In for Automatic Label Placement Using Simulated Annealing”. University of Berkeley, USA. http://github.com/tinker10/D3-Labeler.

Weichselbraun, A., Gindl, S., & Scharl, A. (2013). “Extracting and grounding context-aware sentiment lexicons”. IEEE Intelligent Systems, 28(2), 39-46.

Weichselbraun, A., Gindl, S., & Scharl, A. (2014). “Enriching semantic knowledge bases for opinion mining in big data applications”. Knowledge-Based Systems, 69, 78-85.

Weichselbraun, A., Streiff, D., & Scharl, A. (2015). “Consolidating Heterogeneous Enterprise Data for Named Entity Linking and Web Intelligence”. International Journal on Artificial Intelligence Tools, 24(2), 1540008.

Appendix A: Visualization Workflow

Building on best-practice examples reported in the literature, we adopted the following *workflow* for collecting, processing and visualizing statistical data as part of big data applications:

- **Requirement Analysis** includes the collection of user stories related to the data sets to be integrated, and the identification of appropriate visualizations;
- **Discovery.** If a partner does not have the data in the needed format, we help with selecting the datasets and the dimensions that need to be analyzed, aggregated or augmented in order to fit particular visualization scenarios.
- **Alignment** generally refers to converting the new datasets into the format required by the Statistical Data API (see Appendix C) taking into account granularity, naming conventions, geolocation and other factors.
- **Indicator Storage and Retrieval.** New datasets are stored after being aligned to match the API. Effective indexing strategies based on established platforms such as Elasticsearch[23] and Lucene[24] are essential building blocks of our big data applications (see Section “Index Strategy and Deployment”).
- **Transformation** is rarely needed, as data provided via the API can be directly visualized. The transformation step can be seen as a first part of the data and view specification (filter, derive) step of Heer and Shneiderman (2012), although derive tasks can also appear in subsequent steps.
- **Visualization.** The basic building blocks such as line charts, bar charts and donut charts (see “Visual Analytic Components”) tend to be reusable components, which result in visualization grammars that can be considered the second part of the data and view specification step (visualize, sort).
- **Interaction.** An interaction layer that includes zooming, panning and synchronization mechanisms is usually built on top of the visualization layer, corresponding to the view manipulation step of Heer and Shneiderman (2012), and is the focus of D1.3: ASAP Dashboard.
- **Reuse** can happen on multiple levels, from indicators to specific charts or the entire platform. It should be integral to the design process, corresponding to the process and provenance step of Heer and Shneiderman (2012).

[23] elastic.co

[24] lucene.apache.org

Appendix B: RDF Data Cube Vocabulary

As a W3C Recommendation, the RDF Data Cube Vocabulary (QB)[25] is supported by both industry and academia. It has already gained widespread acceptance judged by the increasing number of statistical datasets published using this vocabulary.

QB is based on a cube model that is compatible with SDMX (Statistical Data and Metadata Exchange), designed to be generic and suitable for publishing various types of multidimensional datasets. The basic building blocks of the cube model are measures, dimensions and attributes, collectively referred to as components:

- *Measure components* describe things or phenomena that are being observed; typically used for measurements – e.g. number of mobile phone calls.
- *Dimension components* specify variables that are important when defining an individual observation for a measurement – e.g., time and space.
- *Attributes* help interpret the measured values by specifying the units of measurement, and additional metadata such as the status of the observation – e.g. unit of measurement, estimated.
- *Observations* are the atomic units in a dataset that represent a concrete measured value for a set of concrete dimension values. When the value of a dimension is the same in a large number of observations (for example, the geolocation), it is convenient to group them into a *slice*.
- A *dataset* that contains observations grouped into slices across dimensions constitutes a cube.
- The *Data Structure Document* (DSD) describes each dataset and contains all the required namespaces and components.
- *Code lists or dictionaries* describe the list of entities that are generally repeated through all the datasets from a publisher (countries, units of measurements, etc.).

[25] www.w3.org/TR/vocab-data-cube

Appendix C: API Specification

The following API specification v0.5 outlines how to share data with the webLyzard knowledge repository, and how to access ASAP visualization services. The API specification contains four separate parts, followed by a workflow description.

1. The *Document API* specification describes the mechanism to upload and annotate the IMR Web content of WP8 (and other third-party text documents), and to integrate them into the ASPA visualization pipeline developed in WP6.
2. The *Statistical Data API* supports the upload of statistical data, e.g. the WIND telecommunications data from WP9.
3. The *Search API* returns relevant documents sets based on a search query.
4. The *Visualization API* renders individual visualizations and allows to embed them into third-party Web applications.

The APIs are stateless RESTful APIs accessed via HTTPS. They typically use JSON as input and output formatting, except in the case of RDF Data Cubes and similar structured exchange formats. All API requests need to be authenticated using JSON Web Tokens. Tokens are time-limited and repository-restricted.

As of August 2016, this specification should be considered beta status. While the overall set of feature is stable, specific parts of the specification might still change in line with use case requirements.

Document API

Introduction

The **Document API** describes the programmatic interface for creating, retrieving, updating, deleting (CRUD) documents with a given *document repository*. To use this functionality of the Document API, the user must have a webLyzard document repository configured. Documents in such repositories are identified by a unique (numeric) <identifier> that is generated by the platform when adding new documents. Subsequent document retrievals, updates and deletions should refer to this identifier.

In addition to this repository-only functionality, the document API also supports annotating existing documents using any of the supported webLyzard annotation tools. For document annotation, no repository is required. The Document API provides the following functionality:

1. Adding a document to a given repository
2. Updating an existing document in a given repository
3. Deleting an existing document from a given repository
4. Querying an existing document from a given repository
5. Annotating an existing document without a given repository

Usage of the document API requires an **access token**. For further information on how to obtain such access token, please refer to the section on *Authentication and Authorization*.

Adding Documents to a Repository (Create)

Documents are always added to a specific repository, the data format has to adhere to the webLyzard Document specification (see below). Adding a document will always result in

1. the creation of a new numeric identifier and therefore a new document in the repository, regardless if the URI already exists, and
2. the execution of all annotation steps as defined for the repository – an additional call to `the` Annotate API is therefore *not required*.

As parameters, the Rest API expects content to be provided as either:

- tokenized content, where
  - sentences are provided
  - no content is provided
- a json string content (`text/html`, `text/plain`), where
  - content and content_type are provided
  - no sentences provided

If both plain text and tokenized content or neither are provided, a processing error (4xx) will be returned. For further information on valid document structure to be send to the Document API, please refer to *webLyzard Document Format* section.

To create a new document, send a POST request to the /<repository> API endpoint, with the body of the request containing the document.

```
{
      "repository_id": "repository",
      "title": "document title",
      "uri": "the document's uri",
      "content": "Therefore we could show that \"x>y\" and \"y<z.\".",
      "content_type": "text/plain"
}
```

Listing 1: A minimal document

```
$ curl -H “Authorization: Bearer <access_token>” -H “Content-Type: ap-
plication/json” -d @document.json -XPOST
https://api.weblyzard.com/0.1/documents/<repository>
```

Listing 2: Adding a document ‘document.json’ to a webLyzard repository via the Document API

If the document has been successfully stored, the server responds with a “201 Created” status code and the “Location” header field contains the unique <identifier> created for this document.

```
HTTP/1.1 201 Created
Location:
https://api.weblyzard.com/0.1/documents/<repository>/<identifier>
Content-Type: application/json; charset=UTF-8
Content-Length: ...

{“created”:true, “_id”:”<identifier>”}
```

Listing 3: REST response from the Document API (document add)

In case of error, the server will return one of multiple error codes and a description of the error. A 4xx error will be returned in case the request is malformed (e.g. “400 Bad Request”) or the user does not have the appropriate access rights (e.g. “403 Forbidden”). A 5xx error will be returned if processing on the server failed.

Retrieving Documents

The most recent version of a document can be retrieved by sending a GET request to the <identifier> of the document:

```
$ curl -H “Authorization: Bearer <access_token>” -XGET
‘https://api.weblyzard.com/0.1/documents/<repository>/<identifier>’
```

Listing 4: Retrieving a document from a webLyzard repository via the Document API

The server responds with a “200 OK” status code and the JSON representation of the document (as specified by the webLyzard document specification):

```
HTTP/1.1 200 OK
Content-Type: application/json; charset=UTF-8
Content-Length: …

{“_id”:”<identifier>”, “repository”:”<repository>”, “title”:...}
```

Listing 5: REST response from the Document API (document retrieve)

If the document does not exist, the server sends a “404 Not Found” response.

Updating Documents

Documents can be overwritten with a newer version of the same document.

```
$ curl -H “Authorization: Bearer <access_token>” -H “Content-Type: ap-
plication/json” -d @document.json -XPUT
https://api.weblyzard.com/0.1/documents/<repository>/<identifier>
```

Listing 6: Updating a document in a webLyzard repository via the Document API

For further information on valid document structure to be send to the Document API, please refer to *webLyzard Document Format* section. On success, the server responds with a “200 OK” status code:

```
HTTP/1.1 200 OK
Content-Type: application/json; charset=UTF-8
Content-Length: …

{“created”:false,”updated”:true,”_id”:”<identifier>”}
```

Listing 7: REST response from the Document API (document update)

If there is no document referenced by <identifier> available, the server will respond with a “400 Bad Request” error and the document should be added using the syntax for adding documents; i.e., <identifier> is always created by the server and cannot be set arbitrarily by the client.

Deleting Documents

Documents can be deleted by issuing a DELETE request on the identifier of the document:

```
$ curl -H “Authorization: Bearer <access_token>” -XDELETE
‘https://api.weblyzard.com/0.1/documents/<repository>/<identifier>’
```

Listing 8: Deleting a document from a webLyzard repository via the Document API

On success, the server responds with a “200 OK” status code:

```
HTTP/1.1 200 OK
Content-Type: application/json; charset=UTF-8
Content-Length: …

{“deleted”:true,”_id”:”<identifier>”}
```

Listing 9: REST response from the Document API (document delete)

Annotating Documents

Instead of storing and annotating a document to a repository, users can request to have a document annotated only – without permanently storing the document in a repository.

The webLyzard Document API currently supports the following document annotations:

1. **`sentiment`**, extracts document and sentence level polarity from a document (also runs sentence tokenization and POS tagging as a prerequisite, if not provided by the user)
2. **`namedentities`**, identifies named entities in a document using our Named Entity Recognition (NER) tool, *Recognyze*.

For information on valid document structures to be send to the annotation service, please refer to the *webLyzard Document Format* section.

```
$ curl -H “Authorization: Bearer <access_token>” -H “Content-Type: ap-
plication/json” -d @document.json -XPOST
https://api.weblyzard.com/0.1/annotate
```

Listing 10: Annotating a document via the Document API

If successful, the server responds with a “200 OK” response code and returns the annotated document:

```
HTTP/1.1 200 OK
Content-Type: application/json; charset=UTF-8
Content-Length: …

{...}
```

Listing 11: REST response from the Annotate API (document annotate)

In case of error, the server will return one of multiple error codes and a description of the error. A 4xx error will be returned in case the request is malformed (e.g. “400 Bad Request”) or the user does not have the appropriate access rights (e.g. “403 Forbidden”). A 5xx error will be returned if processing on the server failed.

To add specific annotations only, the annotation type can be included in the request:

```
$ curl -H “Authorization: Bearer <access_token>” -d @document.json -
XPOST http://api.weblyzard.com/0.1/annotate/sentiment

$ curl -H “Authorization: Bearer <access_token>” -d @document.json -
XPOST http://api.weblyzard.com/0.1/annotate/sentiment+namedentities
```

Listing 12: Annotating a document via the Annotate API with different workflows

Statistical Data API

Introduction

The Statistical Data API is used to insert statistical data into the webLyzard repositories in order to visualize it through our portals or stand-alone interfaces.

The terminology used in this document reflects the one used in statistical data processing. An **observation** is a special type of document that corresponds to a datapoint in a visualization (or a datapoint in a time series). An **indicator** (sometimes referred to as dataset) is a collection of such observations that share the same attributes. All observations need to include data about their provenance (publisher, date, etc.).

The main object consumed by this API is **Observation**.

To use the functionality provided by this API, the user will need to have at least one *webLyzard statistical repository* configured in which it will be able to upload up to 100 million observations.

If the user plans to upload multiple indicators / datasets in the same repository, the user will need to assign *unique identifiers across all the indicators / datasets* that will be stored in this repository (e.g., adding the short name of the indicator in front of the identifier is a quick method to create unique identifiers within a repository).

If the user plans to upload multiple large datasets (e.g., datasets bigger than 100 million records or bigger than 2 TB) or has different needs in terms of how to deploy the API for certain projects (e.g., production and dev/test environments) it is advisable to contact us and discuss such use cases as early as possible.

This API was designed to accommodate time series and provides CRUD functionality for handling statistical data. Currently the following operations are supported:

1. Adding an observation to a given repository.
2. Retrieving an observation from a given repository.
3. Updating an observation from a given repository.
4. Deleting an observation from a given repository.

Currently it is possible to load entire datasets on the fly via custom Python or Java scripts. Next version of the API will also include the ability to bulk load data sets on the fly.

Adding Observations (Create)

An observation is similar to a document consumed by the Document API, but has different fields. Each observation has to adhere strictly to the webLyzard Statistical Data specification (see below).

Adding a new observation will result in:

1. Pushing the data in the Statistical Data repository configured for the respective user
2. Execution of additional calls if other data processing steps are defined.

To create a new observation, send a POST request to the /<repository> API endpoint, with the body of the request containing the observation. While location is not a required field, it is generally expected that an observation will have an associated location if the use case requires visualizing that observation on a map.

```
{
        "_id": "example_1",
        "uri": "http://example.com/example-1",
        "added_date": "2014-09-10T15:01:48.623816",
        "date": "2004-01-01T00:00:00",
        "indicator_id": "esairtrans2",
        "indicator_name": "ES Air Trans 2",
        "value": "1000"
}
```

Listing 13: A minimal observation.

```
$ curl -XPOST
‘https://api.weblyzard.com/0.1/observations/<repository_name>/<indicator
_id>’ -d ‘{
  “_id” : “...”,
  “uri” : “...”,
  …
}’
```

Listing 14: Creating an observation with the Statistical API

If the observation has been successfully stored, the server responds with a “201 Created” status code and the “Location” header field contains the unique identifier created for this document.

```
HTTP/1.1 201 Created
Location:
https://api.weblyzard.com/0.1/observations/<repository_name>/<indicator_
id>/<observation_id>
Content-Type: application/json; charset=UTF-8
Content-Length: ...

{“created”:true, “_id”:”<identifier>”, “version”:”<version>”}
```

Listing 15: Response from the REST API that indicates an observation was created

In case of error, the server will return one of multiple error codes and a description of the error. A 4xx error will be returned in case the request is malformed (e.g. “400 Bad Request”) or the user does not have the appropriate access rights (e.g. “403 Forbidden”). A 5xx error will be returned if processing on the server failed. In case a dataset or indicator is not complete, you can add the missing observations later.

Retrieving Observations

The most recent version of an observation can be retrieved by sending a GET request to the <identifier> of the document:

```
$ curl -XGET
‘https://api.weblyzard.com/0.1/observations/<repository_name>/<indicator
_id>/<observation_id>’
```

Listing 16 Retrieving latest version of an observation

The server responds with a “200 OK” status code and the JSON representation of the observation (as specified by the webLyzard Statistical Data specification):

```
HTTP/1.1 200 OK
Content-Type: application/json; charset=UTF-8
Content-Length: …

{“_id”:”<identifier>”, “repository”:”<repository_name>”,...}
```

Listing 17: Observation retrieved successfully

If the observation does not exist, the server sends a “404 Not Found” response.

Updating Observations

Similar to updating a document in the Document API. Observations can be updated (overwritten) with a newer version of the same observation.

```
$ curl -XPUT
‘https://api.weblyzard.com/0.1/observations/<repository_name>/<indicator
_id>/<observation_id>’ -d ‘{
  “uri” : “...”,
  “content-type” : “...
  …
}’
```

Listing 18: Updating an observation via the API

On success, the server responds with a “200 OK” status code:

```
HTTP/1.1 200 OK
Content-Type: application/json; charset=UTF-8
Content-Length: …

{“creat-
ed”:false,”updated”:true,”_id”:”<identifier>”,”version”:”<version>”}
```

Listing 19: The update was successful

If there is no observation referenced by <identifier> available, the server will respond with a “400 Bad Request” error and the observation should be added using the syntax for adding observations (i.e. <identifier> is always created by the server and cannot be set arbitrarily by the client).

Deleting Observations

Similar to deleting a document in the Document API.
Observations (either a specific version or all versions of an observation) can be deleted by issuing a DELETE request on the identifier of the observation:

```
$ curl -XDELETE
‘https://api.weblyzard.com/0.1/observations/<repository_name>/<indicator
_id>/<observation_id>’
```

Listing 20: Deleting observations

On success, the server responds with a “200 OK” status code:

```
HTTP/1.1 200 OK
Content-Type: application/json; charset=UTF-8
Content-Length: …

{“deleted”:true,”_id”:”<observation_id>”}
```

Listing 21: Delete operation was successful

Search API

Clients need to send `Content-Type:application/json` and `Accept:application/json` headers.

General query structure

A search query conforms to the following JSON document:

```
{
 "sources" : [],
 "fields" : [],
 "query" : {},
 "filter" : {},
 "count" : 10,
 "offset" : 0,
 "ranking" : {}
}
```

Search	
Request Fields	
`source`	A list of strings, specifying the set of sources the query should be run against.
`fields`	A list of strings, specifying which fields should be returned for each document. Possible values: contentid the internal content id of the document title the title of the document url the original URL of the document summary a summary of the full text content of the document snippet a short snippet of content around the best query match in the document quote a longer snippet of content around the best query match in the document sourceid an internal identifier for the source of the document

`query`	Each query object can contain one of the following types of query: bool a Boolean query supported query types: bool title search for text matches in the title supported query types: phrase, regexp, term text search for text matches in the full text supported query types: phrase, regexp, term date limit search results to a date range supported query types: range sentiment limit search results to a sentiment range supported query types: range url limit search results to matching URLs supported query types: regexp, term, wildcard
`filter`	The filter object supports the same syntax as the query object, but does not affect search result ranking. Try to use filters as much as possible to speed up query times (e.g. date and sentiment should always be put into a filter, as it makes no sense to rank based on these) and only use queries for fulltext queries that should affect the order of documents.
`count`	The number of matches to return.
`offset`	Offset for the documents to return.
`ranking`	Allows to influence the order of search results in the response.
`boost`	A list of boosting queries to influence search result order.

Query Types

Bool Query

A Boolean query combines any other queries into a single, aggregated query:

```
"bool" : {
       "must" : [],
       "should" : [],
       "must_not: []
}
```

must – each of the sub-queries must match.
should – any of the sub-queries should match.
must_not - none of the sub-queries must match.

Phrase Query

A phrase query tries to match the given string as a phrase in the field:

```
"text" : {
 "phrase" : "climate change
}
```

Regexp Query

A regexp query tries to match the given string as a regular expression in the field:

```
"title" : {
 "regexp" : "climate( |-)change
}
```

Allowed regular expression operators:
`()` grouping
`|` alternatives
`?` optional parts

Term Query

Match the given string to the full content of the field:

```
"title" : {
 "term" : "Media Watch on Climate Change"
}
```

Wildcard Query

Match the given string to the full content of the field, supporting wildcards:

```
"url" : {
 "wildcard" : "www.google.com/*"
}
```

where `*` matches any number of characters and `?` matches a single character.

Range Query

A query that supports matching values in a range:

```
"sentiment" : {
 "lt" : 0,
 "gt" : -0.5
}
```

```
"date" : {
 "gte" : "2014-01-01",
 "lte" : "2014-12-31"
}
```

`eq` – value equals the given value

`lt` – value is less than the given value

`gt` – value is greater than the given value

`lte` – value is less than or equal to the given value

`gte` – value is greater than or equal to the given value

Supported date formats:

yyyy-MM-dd
yyyyMMdd
dd-MM-yyyy
ddMMyyyy

Boosting Queries

Query ranking can be influenced by setting boosting queries, where each boosting query has the following attributes:

- **query** ranking weights for documents matching the query are changed
- **mode** score replacement mode (supported values: `replace`, `sum`, `mult`)
- **boost** weight that influences the score.

Response

```
{
 "more" : true|false,
 "hits": [],
 "total": 0
}
```

`more` – `true` if more hits than the returned exist
`hits` -a list of documents
`total` – the total number of documents matching the query

Note: if only document counts need to be retrieved, put all queries inside the filter attribute and set count to 0 for best performance.

Examples

```
{
 "sources" : ["climate2_media"],
 "fields" : ["title", "summary"],
 "query" : {
   "text" : {
     "phrase" : "climate change"
   }
 },
 "filter" : {
   "date" : {
     "gte" : "2015-01-01",
     "lte" : "2015-01-31"
   }
 },
 "count" : 100,
 "offset" : 0,
 "ranking" : {}
}
```

Example 1: Search for 100 documents matching "climate change" as a phrase in the full text in January 2015 in News Media in the Media Watch on Climate Change, returning title and summary

```
{
 "sources" : ["climate2_media"],
 "fields" : ["title", "summary"],
 "query" : {
   "bool" : {
     "must" : [{
       "text" : {
         "phrase" : "climate change"
       }
     }, {
       "title" : {
         "phrase" : "climate change"
       }
     }]
   }
 },
 "filter" : {
   "bool" : {
     "must" : [{
       "date" : {
         "gte" : "2015-01-01",
         "lte" : "2015-01-31"
       }
     }, {
       "sentiment" : {
         "lt" : 0
       }
     }]
   }
 },
 "count" : 100,
 "offset" : 0,
 "ranking" : {}
}
```

Example 2: Search for 100 negative documents matching "climate change" as a phrase in the full text and in the title in January 2015 in News Media in the Media Watch on Climate Change, returning title and summary

```
{
 "sources" : ["climate2_media"],
 "fields" : ["title", "url"],
 "query" : {
   "text" : {
     "phrase" : "climate change"
   }
 },
 "filter" : {
   "date" : {
     "gte" : "2015-01-01",
     "lte" : "2015-01-31"
   }
 },
 "count" : 100,
 "offset" : 0,
 "ranking" : {
   "boost" : [{
     "query" : {
       "url" : {
         "wildcard" : "*.cnn.com/*"
       }
     },
     "mode" : "mult",
     "boost" : 10
   }]
 }
}
```

Example 3: Search for 100 documents matching "climate change" as a phrase in the full text in January 2015 in News Media in the Media Watch on Climate Change, returning title and url, favoring matches on CNN

Visualization API

The webLyzard dashboard offers a feature-rich and customizable solution for visual analytics, semantic search and Web intelligence applications. To support use cases that require a more granular approach, the webLyzard Visualization API enables the integration of distinct portal components into third-party Web applications.

Version 1 uses `<iframe>` tags to embed these components. While this approach ensures ease of use and widespread compatibility across platforms, it also comes with shortcomings if a deeper integration is desired. This will be addressed by additional features in future versions of the API, complementing (but not replacing) the `<iframe>` approach.

`<iframe>`	
`Attribute` The iframe should be provided with all necessary attributes, width and height could be preset to the desired dimensions, similar to the approach of e.g. YouTube.	`Fields`
`width`	`int` width in pixels
`height`	`int` height in pixels
`frameborder`	`int` should always be `0`
`seamless`	`Boolean` `true` if you want to inherit styles from the parent window and open links in the parent window
`sandbox`	`string` not required, but if present a value of `'allow-same-origin allow-scripts'` is required
`src`	`string` see next section “URL Schema”

URL Schema	
`/embed/:view`	`view:string` `:view` determines the desired map to be shown in the iframe. One of `documents`, `quotes`, `frequency`, `distribution`, `tags`, `keywords`, `cluster` or `geo`
`/api/:view?format=:format`	`view:string, format:string` Similar to above but more generic by using a query parameter to determine the format: one of `html`, `xml`, `xls`, `csv`, `png`, `svg` or `json`

`Query-String Schema` Additional (optional) query-strings allow the configuration of the embedded visualization.	
`search`	`string` optionally this supports a comma separated list of search strings, which will be treated as individual searches to support multiple lines or slices in the trend chart and donut chart
`begindate`	`string`
`enddate`	`string`
`topics`	`string` a comma separated list of unique numerical IDs, representing predefined topic definitions
`indicator`	`String` a comma separated list of unique string IDs, representing predefined statistical indicators

`Examples`
<!-- The tag cloud using the default search term --> <iframe width='400' height='600' src='/embed/tags' frameborder='0'>
<!-- The geo map using a custom search term --> <iframe width='400' height='600' src='/embed/geo?search=fracking' frameborder='0'>

Authentication and Authorization

Authentication and authorization is handled using JSON Web Tokens (JWT). Until tokens are issued using the global webLyzard login server, new tokens can be obtained using the `/token` API endpoint using Basic Authentication.
A token is valid for 8 hours, after which the token will be rejected by the API and a new token must be generated.

Obtaining a new token

To obtain a new token, do a GET request to the `/token` endpoint:

```
$ curl -i -u <user>:<pass> https://api.weblyzard.com/0.1/token
```

The server responds with the issued token for the user:

```
HTTP/1.1 200 OK
Date: Tue, 17 Nov 2015 12:43:10 GMT
Server: Apache/2.4.7 (Ubuntu)
Content-Length: 626
Connection: close

eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzUxMiJ9.eyJwZXJtaX...
```

Calling API methods using the obtained token

All API calls must be authenticated using a valid token (see above). Pass the token using the "`Authorization: Bearer`" request header:

```
$ curl -i -H "Authorization: Bearer
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzUxMiJ9.eyJwZXJtaX..."
https://api.weblyzard.com/0.1/documents/test/12345
```

Document Format

The webLyzard document format consists of three related object structures: the *Document*, the *Sentence*, and the *Annotation*. These three data structures strongly depend on each other, but only *Document* is essential.

The *Document* structure models a single document to be uploaded. It provides the system with the basic information required to process this single document. Fully tokenized and/or annotated documents can be provided via the *Annotation* (surface form annotations such as Named Entity (NE) and target sentiment) and the *Sentence* (tokenization/ POS/sentiment) formats as documented below.

Accepted document encoding is limited to UTF-8.

`Document`	
Required Fields	
`uri`	`string` the unique identifier of the document, e.g. a URL. Must be a valid URI.
`title`	`string` a title string
Optional Fields	
`repository_id`	`string` a unique ID consisting of a repository name + fqdn of the content provider, source or project – for example: mobile.asap-fp7.eu, media.ecoresearch.net, or social.weblyzard.com.
`language_id`	`string` `ISO language identifier. Supported are ['en','es','fr','de']`
`content_type`	`string` only required if **`content`** is provided, specifies how the content should be interpreted, supported are `text/html`, `text/plain`
`content`	`variable` the document content as json string, with **`content_type`** specifying the respective content format. If `content` is provided, then **`sentences`** must not be provided. Providing both `content` and `sentences` will result in an error. Providing `content` without `content_type` will result in an error.
`sentences`	`list` an ordered list of tokenized webLyzard **`sentence`** objects. If `sentences` are not provided, **`content`** and **`content_type`** must be provided.

annotations	list a list object of webLyzard **annotation**. Currently supported by the visualization components are sentiment and named_entity types, but may be used also for other annotations types (such as rumours, opinion targets, etc.).
meta_data	dict a dictionary describing arbitrary document metadata that provides additional document-level information, for example: author — the author of the document published_date — a date string determining when a document was published (and therefore when it will be visible in the portal). If no published_date is provided, we will try to extract one from the content. If this fails, the submission date of the document is used as the published_date polarity — document-level sentiment polarity document_linkage — 1) 'refers-to', n-to-n, for retweets, quotes, etc... 2) 'child-of', 1-to-1, to model nested conversations (threaded dialogues) 3) 'part-of' , n-to-n, document belongs to e.g. story cluster / other collection
features	dict A dictionary describing arbitrary data as key value-pairs, which complements the more well-defined meta_data field. These key value-pairs are disregarded in the visual analytics dashboard, unless custom frontend functions were developed to process them.
relations	dict A dictionary describing arbitrary document-to-document relations as key-value pairs. Document relations do not have any impact in the portal unless explicitly requested.

Examples

```
{
    "repository_id": "media.ecoresearch.net",
    "uri": "http: //www.bbc.com/news/science-environment-33524589",
    "content_type": "text/plain",
    "title": "New Horizons: Nasa spacecraft speeds past Pluto",
    "content": " Nasa’s spacecraft speeds past Pluto",
    "meta_data": {
        "author": "Jonathan Amos"
    }
}
```

```
{
       "repository_id": "media.ecoresearch.net",
       "uri": "http: //www.bbc.com/news/science-environment-33524589",
       "title": "New Horizons: Nasa’s spacecraft speeds past Pluto",
       "sentences": [
       {
              "id": "595f44fec1e92a71d3e9e77456ba80d1",
              "value": "New Horizons: Nasa’s spacecraft speeds past Pluto",
              "is_title": "TRUE",
              "pos_list": "NN NN : \' NN : NN CC JJ NN . \'",
              "tok_list": "0,2 3,19 19,20 21,22 22,33 33,34 35,42 43,46 47,55
56,62 62,63 63,64",
              "sentence_number": 0,
              "polarity": -0.783
       }
       ],
       "annotations": [
       {
              "start": 12,
              "end": 16,
              "sentence": "595f44fec1e92a71d3e9e77456ba80d1",
              "surface_form": "Nasas",
              "annotation_type": "OrganizationEntity",
              "key": "http://dbpedia.org/page/Nasa"
       },
       {
              "start": 40,
              "end": 44,
              "sentence": "595f44fec1e92a71d3e9e77456ba80d1",
              "surface_form": "Pluto",
              "key": "http://dbpedia.org/page/Pluto",
              "annotation_type": "GeoEntity"
       }
       ],
       "meta_data": {
           "polarity": "0.342",
           "published_date": "2015-07-14"
       }
    }
}
```


Sentence	
Required Fields	
id	string the unique identifier of the sentence, i.e. a sentence hash (md5)

`value`	`string` the sentence text
`pos_list`	`string` a whitespace separated list of part-of-speech tags (POS), one per token. Currently supported POS by language are specified at weblyzard-api.readthedocs.org/en/latest/weblyzard_api.data_format.pos-tags.html
`tok_list`	`string` a whitespace separated list of sentence tokens (words), encoded as space-separated string of comma-separated sentence offset tuples: start_offset,end_offset, e.g. `"0,2 3,19"`
Optional Fields	
`is_title`	`Boolean` is the sentence part of the title, defaults to False If the document-level attribute “title” is also set, the value of this sentence must match that attribute.
`dep_tree`	`string` a whitespace separated list of pointers to the parent of a token in the dependency tree. -1 denotes the root node.
`sentence_number`	`int` 0-based sentence sequence number, e.g. the index of a sentence in the list of all document sentences
`paragraph_number`	`int` 0-based paragraph sequence number, e.g. the index of a paragraph in the list of all document paragraphs
`polarity`	`float` sentence-level sentiment polarity as floating point in range [-0..1]
`polarity_class`	`string` sentence-level sentiment polarity class, with possible values [‘positive’, ‘negative’, ‘neutral’]
Examples	

```
{
    "id": "595f44fec1e92a71d3e9e77456ba80d1",
    "value": "New Horizons: Nasa’s spacecraft speeds past Pluto.",
    "is_title": False,
    "pos_list": "NNP NNP : NNP POS NN NNS IN NNP .",
    "tok_list": "0,3 4,12 12,13 14,18 18,20 21,31 32,38 39,43 44,49 49,50",
    "sentence_number": 0,
    "polarity": -0.783,
    "polarity_class": "negative"
}
```

```
{
    "id": "595f44fec1e92a71d3e9e77456ba80d1",
    "value": "New Horizons: Nasa’s spacecraft speeds past Pluto."
}
```

Annotation	
Required Fields	
`start`	`int` the start offset of the annotation, relative to the absolute document content (not tokenized).
`end`	`int` the end offset of the annotation, relative to the absolute document content (not tokenized).
`surface_form`	`string` the surface form of the annotation (e.g. how the annotation actually appears in the document)
`annotation_type`	`string` the type of the annotation. Supported by the visualization components are `Sentiment`, `GeoEntity`, `PersonEntity`, `OrganizationEntity`. Arbitrary other sentence-level annotations are allowed, but not currently supported by the visualization components.
Optional Fields	
`key`	`string` reference key, e.g. Linked Open Data (LOD)
`sentence_id`	`string` the **id** of the **`sentence`** object to which the annotation’s start and end positions refer to. If no sentence id is specified, the annotation positions are applied on the document level.
`display_name`	`string` searchable field in the portal
`polarity`	`float` sentence-level sentiment polarity as floating point in range [-0..1]
`polarity_class`	`string` document-level sentiment polarity, with possible values [`‘positive’`, `‘negative’`, `‘neutral’`]. Requires `annotation_type` to be `sentiment`.

properties	dict a dictionary of additional properties associated with the annotation. The expected key value tuples in the properties depend on the type of the entity defined on the webLyzard **document** level (e.g. the key to the annotation list). Supported properties by the portal are: lat, long, population, birth_date, abstract

Examples

```
{
    "start": 87,
    "end": 92,
    "sentence": "595f44fec1e92a71d3e9e77456ba80d1",
    "surface_form": "Apple",
    "key": "http://dbpedia.org/page/Apple_Inc",
    "annotation_type": "OrganizationEntity",
    "display_name": "Apple Incorporated",
    "properties": {
        "founders": "Steve Jobs,Steve Wozniak,Ronald Wayne"
    }
}
```

```
{
    "start": 12,
    "end": 14,
    "sentence": "595f44fec1e92a71d3e9e77456ba80d1",
    "surface_form": "USA",
    "key": "http://dbpedia.org/page/United_States",
    "annotation_type": "GeoEntity",
    "display_name": "U.S.A",
    "properties": {
        "population": "318.900.000",
        "lat": "100.0",
        "long": "30.0"
    }
}
```

```
{
    "start": 12,
    "end": 14,
    "sentence": "595f44fec1e92a71d3e9e77456ba80d1",
    "surface_form": "USA",
    "polarity": 0.655,
    "annotation_type": "Sentiment"
}
```

```
{
    "start": 12,
    "end": 14,
    "surface_form": "USA",
    "annotation_type": "GeoEntity"
}
```

Statistical Data Format

Datasets from partners and clients (e.g., phone call data, tourism statistics), open government sources, or international organizations such as Eurostat and the World Bank are often statistical in nature – each data point corresponds to an observation with timestamp, value, unit of measurement and several other attributes or dimensions. Such data comes in multiple formats including JSON, RDF (Classic RDF, SDMX or RDF Data Cube), and XML.

The following JSON format based on the popular RDF Data Cube (QB) standard ingests such heterogeneous data into into the visualization pipeline. Each data point can be considered the equivalent of a document in the Document API, but the Statistical Data API is separate – there are no dependencies among data types.

Statistical Data	
Required Fields	
`_id`	`string` the unique identifier of an observation. *All identifiers need to be considered unique per partner/client and not per indicator.*
`uri`	`string` the unique identifier of the document, e.g. a URL of the respective observation Example: http://eurostat.linked-statistics.org/data/ttr00012#A,PASS,CZ,2004
`added_date`	`date` the date when the observation was added to the index – current date
`date`	`date` the observation date Format: 2004-01-01T00:00:00
`indicator_id`	`string` a short string representing a unique identifier of the indicator without any whitespaces. Indicator_id should be similar to variable names. Example: esairtrans – might represent an indicator_id for the indicator Eurostat Air Transport (see the next field).
`indicator_name`	`string` a short or long name of the indicator. This is the name that will be displayed for the respective indicator in our portals so it needs to give a clear indication of the provenance and significance of the data. in case the data does not come from a statistical database (therefore it is not organized around indicators), this field should

	contain a short descriptive name of what type of data we have (e.g., European phone calls, Italian tourism data). In general we recommend the following convention for naming indicators: provenanceCode + short description. This field can contain whitespaces. Indicator_name should be similar to label names, as it is in fact a label for the indicator name. In certain cases it is also allowed to have indicator_name exactly the same as the indicator_id, though of course it will not include spaces in such a scenario. Example: *ES Air Trans* – stands for *Eurostat (ES) Air Transport* – *ES* designates provenance (Eurostat, ES) and *Air Trans* is a short meaningful description of what the indicator represents (*Air Transport*)
`value`	`float` the observation value
`repository_id`	`string` the unique identifier of a repository from webLyzard where this information is stored.
Optional Fields	
`year, month, day, hour`	`string` you can use shortcuts for year, month, day, hour to ease the search
`location_id`	`string` If location information is available as GeoJSON and in other formats, it is recommended to use this field and add the id of the location in it (the region id, or the GeoJSON file id, etc.).
`target_type`	`string` the type of geographical unit that we find in the target. In general the locations that are not geo political / administrative (city, country, region) should be marked as points of interests (poi). *Accepted values: {city, country, region, poi}*
`target_poi_type`	`string` the type of poi in order to distinguish e.g. landmarks, historical places, monuments, parks, office buildings, etc.
`target_country`	`string` 2 letter ISO code of the target country

target_location	geolocation
	the geolocation of the target – can include name and the geo co-ordinates (lat and long).
	
	Example:
	{
	"name": "New Zealand",
	"point": {
	"lat": "-42.0",
	"long": "174.0"
	}
	}

source_type	similar to target_type
source_poi_type	similar to target_poi_type
source_country	similar to target_country
source_location	similar to target_location
producer	string the name of the institution who published the data Example: Eurostat, World Bank, etc.
frequency	string the sampling frequency Example: {year, month, day}
description	string a description of the indicator
unit_of_measurement	string while many of the datasets published today do not contain *a unit of measurement*, it would be recommended to add such a field if this information is present in your dataset
type	string the default type is *observation*; you could use this field if you also decide to save other values like *averages, clustering results* for observation groups and not just the raw observations
Examples	
EUROSTAT data (required fields marked with bold) Current example does not follow identifier convention. { **"_id": "111",** **"uri": "http://eurostat.linked-statistics.org/data/ttr00012#A,PASS,CZ,2004",** **"added_date": "2014-09-10T15:01:48.623816",**	

```
    "date": "2004-01-01T00:00:00",
    "indicator_id": "esairtrans",
    "indicator_name": "ES Air Trans",
    "value": 9950314,
    "repository_id": "eurostat",
    "description": "Air transport of passengers",
    "producer": "Eurostat",
    "sample": "tourism_statistics",
    "frequency": "year",
    "year": 2004,
    "target_country": "CZ",
    "target_type": "country",
    "target_location": [
        {
            "name": "Czech Republic",
            "point": {
                "lat": 49.75,
                "long": 15.0
            }
        }
    ],
    "observation_type": "observation"
}
```


WORLD BANK data EUROSTAT data
(required fields with bold)
Current example does not follow identifier convention.


```
{
    "_id": “11102",
    "uri": "http://worldbank.270a.info/dataset/world-bank-
indicators/PA.NUS.FCRF/NZ/1982",
    "added_date": "2014-09-10T15:00:02.294083",
    "date": "1982-01-01T00:00:00",
    "indicator_id": "wbexchgrate",
    "indicator_name": "WB Exchange rate",
    "value": 1.33260833233333,
    "repository_id": "worldbank",
    "description": "Official exchange rate (LCU per US$, period average)",
    "producer": "World Bank",
    "sample": "tourism_statistics",
    "frequency": "year",
    "year": 1982,
    "target_country": "NZ",
    "target_type": "country",
    "target_location": [
        {
         "name": "New Zealand",
         "point": {
             "lat": -42,
             "lon": 174
             }
         }
   ],
   "observation_type": "observation"
}
```

FP7 Project ASAP
Adaptable Scalable Analytics Platform

End of ASAP
D6.3 InfoViz Services v2

WP 6 – Information Visualization
webLyzard technology

Nature: Report
Dissemination: Public