Intelligent Tools for Policy Design

Deliverable 3.10

Final Software Design Description

Project Reference No.	287119
Deliverable No.	D 3.10
Relevant workpackage:	WP 3
Nature:	Report
Dissemination Level:	Public
Document version:	FINAL
Editor(s):	Nikolaus Rumm, Robert Thaler, Bernhard Ortner, Andreas Noack, Michael Kaufmann, Peter Orgon
Contributors:	Peter Sonntagbauer, Susanne Sonntagbauer, Mario Neumann, Hakan Kagitcioglu, Peter Mairhofer, Anton Jessner, Alexander Kamenicky
Document description:	The objective of this document is to describe the core platform's final architecture, supplemented by selected design details. The document's structure is aligned with that of its predecessor D3.2 and we tried to avoid duplication, referring to D3.2 where necessary and focusing on the changes that were introduced to the system since then.

History

Version	Date	Reason	Prepared / Revised by
0.1	2015-07-02	Initial release (based on D3.2)	Rumm Nikolaus
0.2	2015-07-14	Added the SAP HANA parts	Rumm Nikolaus, Robert Thaler, Bernhard Ortner
0.3	2015-07-22	Reworked the HCP details based on feedback	Rumm Nikolaus
0.4	2015-08-07	Update	Rumm Nikolaus
0.5	2015-08-28	Final release for internal review	Rumm Nikolaus
1.0	2015-08-31	Final release	Rumm Nikolaus

Table of Contents

Management Summary

The objective of the FUPOL project is the development of a new governance model to support the policy design and implementation lifecycle. The innovations are driven by the demand of citizens and political decision makers to support policy domains in urban regions with appropriate ICT technologies. Those policy domains are very important, since more than 80% of the whole population in Europe lives in urban regions and the share will further grow in the future.

Deliverable D3.10 is the final software design description of the FUPOL Core Platform. The FUPOL Core Platform is a central module of the FUPOL System, providing services to the FUPOL users and to the other FUPOL modules:

- Centralized access and account management (security, user management)
- Campaign management (support for research activity)
- Client management (support for multi-client operations)
- Data and knowledge management
- Social media management including content crawling from Twitter, Facebook and other social media sites
- Operational support (services that support the reliable operations of the FUPOL System like logging)
- Integration services (messaging middleware, service coupling, …)

An important note is that this document covers the FUPOL Core Platform, but not the complete FUPOL System. Thus all details mentioned in this document, the architecture and the design focus on the core platform. Interactions with the other FUPOL modules are explained on interface level, but lack any further detail, as these have to be specified for the respective modules separately.

In order to fully understand the FUPOL Core Platform we recommend starting with D3.6 Revised Requirements Specification and Use Cases in order to get an understanding of the system's purpose and the requirements that drive this

architecture. The deliverable D3.11, which was published at the same time as this document, provides an up-to-date description of the final core platform's features as of late August 2015, including the results of the benchmark tests that we performed. This deliverable (D3.10) is based on D3.2.

There are significant dependencies between the content of D3.10 and other deliverables (mainly from WP2/4, WP5 and WP6) which have to be respected to design the FUPOL Core Platform based on the requirements as documented in D3.6 and in this deliverable.

Besides describing the architecture and the design of the final core platform we took the opportunity for some reflection on the design decisions that we made. Most of them are still valid, but with today's knowledge and the technologies, services and products that are available now we'd like to propose changes or enhancements to our approach for the interested reader. Some of them might be implemented during the commercialization of FUPOL.

1 Introduction and Goals

This is the final architecture and design documentation of the FUPOL core platform, based on the project state of late August 2015, just a few months before the project's deadline. We don't expect any changes to the architecture until the project closes in November 2015.

The final system's architecture is still based on the descriptions outlined in D3.2 (Preliminary Software Design Description Prototyp), but we had to revoke some of the assumptions that we made in the project's early stages and of course we adapted to the feedback from pilot cities, scientific and engineering work packages and possible customers.

So overall the system's architecture, based on SOA (service-oriented architecture), was stable and defined a technical system that was able to adapt to the changed requirements throughout the project.

Major design changes included the elimination of the semantic data store (including other semantic web technology) from the system due to severe stability issues and several changes to the way in which the system's modules (text processing, visualization, simulation) interact with each other. All these changes were implemented without significant updates to the architecture.

As outlined before the architecture favoured isolation of concerns and extensibility over performance (one of SOA's principles), which supported the project's team in working locally and integrating the system's modules later, but impacted the performance of some functionality, namely text processing.

We added notes and remarks to this document whenever relevant as a critical review to some of our design decisions and to provide lessons learned for other related projects.

1.1 The Purpose of the Project

Please refer to D3.2 for an introduction of the project's scientific and business background, its goals and its general approach.

1.2 Requirements Overview

For a detailed overview of the requirements of WP3 read the software requirements specification in D3.6 Revised Requirements Specification and Use Cases.

Note that the project's development schedule is driven by user stories as the team has chosen Scrum as their project management framework, but the software requirements specification lists the requirements in form of a concise and consistent document.

1.3 Quality Goals

The system's overall quality must be optimized to meet quality goals. The following table was copied from D3.2, but we repeat it here because it was one of the major driving forces behind the architecture and its understanding is important to get the idea behind some design decisions. Furthermore we added some notes on the goals that explain our experience with them.

Priority	Quality Goal	Rationale
1	Extensibility	Social media is in a state of permanent change. During our project duration it's very likely that i.e. additional social media will enter the market and the architecture must be able to extend the system to integrate these newcomers. The same applies to simulation technologies and products. The system must be extensible to integrate 3rd party simulation products. Notes (2015): extensibility was key to the architecture throughout the project, as we had many changes related to the interaction of modules (text processing, visualization, simulation). In general the coupling of the core platform to text processing and visualization became more important than anticipated in the beginning, while the coupling with the simulation module got looser over time. During the project social media underwent a major change in usage, mostly triggered by the Snowden leaks. While people were careless to provide personal data in social networks and their operators supported this with generous

		access to personal data, the situation has changed significantly. As a consequence we had to remove functionality from the system (i.e. we were unable to access Chinese social media content and Facebook has dropped important functions in their API recently).
2	Adaptability	The system will be operated as a cloud based service and must be able to handle various clients (customers) at the same time providing a virtual partition with exclusive data storage to them. Notes (2015): Again, caused by the Snowden leaks the focus on privacy and data control has increased and this influenced our cloud strategy, which was postponed several times during the project. A pure cloud based system has a much smaller potential for exploitation, as governments face severe limitations in their ability to collect, use and share personal data within their organization. The pilot cities and possible future customers in general preferred on-premise installations over public cloud services for this kind of data. So we had to consider a solution that's capable of public and private cloud operations. This was not anticipated in the early project stages.
3	Accuracy	The system must support the users in generating accurate and useful data describing current trends in social media and precise simulation results. Accuracy in this sense is defined as generating business value, i.e. the system must produce results that are precise and accurate enough to be of use in real-life policy making scenarios.
4	Privacy	In order to be able to reach a high level of user acceptance (especially among the eCitizens) the system

		must protect their privacy with care. Notes (2015): As already mentioned, the privacy concerns have significantly changed during the project (more on the pragmatic level, not so much on the legal, which seems to be much more constraining to the public administrations). We had a permanent struggle between what's technically possible and what the pilot cities would be able/allowed to use. For example we decided to anonymize content very early, but it was very easy to circumvent this obfuscation strategy by just searching for a tweet in Google. So finally we gave it up and the system is now to some extent able to associate content with a user (i.e. to search for content that was authored by some user).
5	Scalability	The pilot scenarios currently cover 5 different clients (customers) but the final product must scale up to a significantly larger number than that (see WP3-79 for details). Notes (2015): the current pilot system stores several ten millions of postings (depending on the client), which was not so much of a limiting factor for the core platform, but the scalability issues were significant with visualization and text processing.
6	Internationali-zation	Our clients (customers) live in different countries using various languages (see WP3-15 for details) including non-european languages like Mandarin. The system must be able to handle them. Notes (2015): we had no problems with European languages, but the processing of Chinese content was a challenge for text processing.

Non-goals

The following quality aspects are considered to be of minor relevance to the system and its architecture and thus won't be adressed explicitly:

Non-goal	Rationale
Look&Feel	As this is a research project it's not a primary quality goal to produce a software system the delivers eye-candy to its users. The system must be usable, implement a consistent and understandable user experience, but not necessarily appear super-attractive. However it was decided that those parts that are exposed to the public (eCitizens) must be implemented in a way that is consistent with the user's expectations of comparable systems. Otherwise we could not attract enough users and would lose a significant number of opinions.

1.4 Stakeholders

A thorough description of this project's stakeholders can be found in D3.6 Revised Requirements Specification.

1.5 The Hands-On Users of the Product

A thorough description of this project's stakeholders, including archtetypes representing them, can be found in D3.6 Revised Requirements Specification.

2 Architecture Constraints

We don't repeat the architectural constraints as outlined in D3.2 here, but add comments on some of them where necessary on the following pages.

2.1 Technical Constraints

2.1.1 Software Requirements

2.1.1.1 Data Structures

The business data model is explained in D3.6 Revised Requirements Specification.

As we decided not to use proprietary GIS data from the pilot cities and instead rely on free (Open Street Map) or commercial (Google Maps) data, the requirements for storing geographical data (i.e. compatibility with INSPIRE) were mostly dropped. The reason for this is that the simulators didn't need the local geographical data that we anticipated in the project's early stages. Finally GIS data in the core platform is used in the following contexts:

- for drawing base maps (backgrounds like city maps)
- for drawing the position of a posting in case we know its coordinates
- for expressing/drawing location-related opinions (opinion maps)

The SIOC/FOAF/DC ontology was used to represent social media data, including relations between eCitizens. This decision was stable throughout the project and the selected ontologies met our requirements.

Slight adaptations to the founding principles of SIOC had to be added in case of microblogging (as there's no concept of "forum" in Twitter), though. Please refer to D3.6 for details on how each social media network's data was matched to the FUPOL ontology. Furthermore the <foaf:Person> was only used as a placeholder for an (observed) account, as we never unified accounts that relate to the same person. So a (natural) person still has several <foaf:Person> representations in the system if the same person uses more than one account. However, this is not an issue for the business requirements that we identified or to those that were raised by the pilot cities.

FUPOL uses several social media sites as a data source for opinions, social relations etc.

When we started the project the social media landscape was different from today's. Web based social media sites like Facebook were in heavy use and instant messaging systems were an idea of the past. Today it seems that the tide is turning and especially the younger people make heavy use of mobile instant messaging solutions like WhatsApp, while Facebook's level of attraction to them is in decline.

We were not able to access social media content from China for political reasons, even that we tried hard for several months.

Anyway, the current core platform's design uses point-to-point connections to the APIs of Facebook and Twitter, which was easy to implement at first, but required some maintenance over time, as these social media sites change their APIs regularly based on changes of their business model. Our observation is that in general access to user generated content is more restricted than it was four years ago. For example Facebook limited access to personal walls (activity streams) in May 2015 and there's no way for us to circumvent this decision, so we had to remove already implememented functionality from the system.

For future projects we'd recommend to avoid the maintenance-effort that is caused by directly talking to the vendors' APIs by using one of the now available social media data aggregators (i.e. Datasift), thus delegating the maintenance of those interfaces to an external party. Furthermore these services aggregate much more data than we could ever do (i.e. some of them have direct access to Twitter's firehose).

2.1.1.2 Software Interfaces

All existing applications that are part of FUPOL are loosely coupled (using an enterprise service bus).

The decision to go the SOA-path was justified by the quality targets as outlined in 1.3 and this decision is still valid, though it imposed several limitations, especially to text processing and – to a lesser extent – to the visualization.

These limitations mostly relate to the overhead that is required to synchronize copies of the crawled content between the core platform, the topic/categorization/summarization system (developed by WP6) and the visualization. The effect of the limitations is decreased performance and scalability.

We introduced a mechanism to cope with them (so that the system still produces acceptable results within reasonable time, i.e. by implementing "forgetfulness"), but we couldn't overcome the principal limitations. The limitations were verified in our benchmark tests (see D3.11) and most database traffic relates to synchronizing the topic/category results with the core platform's data.

2.1.2 System Operations

2.1.2.1 Batch- or Online Operations

FUPOL is an online processing system.

Batch-like operations are part of the system in some specific modules (i.e. the crawler and social media management). This is usually done when time-triggered processing happens. See the use cases in D3.6 Revised Requirements Specification for details on that.

Batch processing is a practical consequence of the limitations that are imposed by SOA and the current HTS algorithm (loose coupling between modules, distributed

data store, discrete training of the HTS model instead of streamed training). For example WP6 originally used a derivative of online-LDA (latent Dirichlet allocation) for topic extraction, which was better suited for processing streaming data than the now-in-use NMF (non-negative matrix factorization) algorithm. The switch between the algorithms had to be performed as LDA produced weak results and NMF was much better in terms of quality of the generated topics. Anyway, the use of NMF meant that the topic model's training had to be triggered at discrete points in time, and as a consequence of this the learned topics were transfered back to the core platform in batches. This imposed significant load both on the text processing module and on the core platform and we suspect that this is one of the triggers of the instability that we observed with Virtuoso (the RDF store that was used in year two and three).

In a future project we would add the text processing module directly to the core platform and thus prevent the overhead that is caused by loosely coupling them. This is the approach that SAP HANA took internally, where the text mining features are directly executed in the in-memory database and thus able to process content in real-time.

The batch processing that is used when learning topics and tagging content with categories caused some confusion for end users, as they expected that a manually created category would immediately be used to tag the content, but instead this happened after some time (i.e. ten minutes). Furthermore it was difficult for the users to understand that the system didn't tag the whole content (i.e. all the past) with their categories. This "forgetfulness" (usually the system considers the previous three months) had to be added to cope with the data volume and one can argue about the size of the timeframe, but not about the necessity of such mechanism.

Later in the project WP6 added the notion of "short term topics" and "long term topics", which is just another view on the same problem.

2.1.3 Programming Requirements

2.1.3.1 Libraries, Frameworks, Components

The core platform provides a web based user interface. There was and still is a large number of frameworks available that support the developers in implementing display and interface logic, including validation.

Such a framework usually provides the following features:

- Separation of concerns (model-view-controller paradigm, where the view and the controller are supported by the web framework)
- Widget repository
- Support for web 2.0 features, especially deep integration of AJAX (using JavaScript libraries)
- Support for client-side validation

We decided to use (server-side) Wicket, which was more advanced and flexible than the other Frameworks that we considered by that time. Unfortunately it was difficult to learn for new team members (that joined the project later), and in a new project we'd not use it again and instead go for one of today's client-side frameworks that is better suited to support i.e. mobile platforms. Anyway, as client-side technologies evolve at a high pace, any decision taken would sound suboptimal after four years.

2.1.4 Methodical Requirements

2.1.4.1 Analysis and Design Methodologies

Analysis and requirements engineering was performed following the Volere requirements engineering process by Suzanne and James Robertson from The Atlantic Systems Guilde.

This requirements structure was adequate and of big value to communicate requirements between cities, the product owner and the developers. The fact that we maintained it in a wiki added flexibility and transparency to the requirements.

On top of that an agile product planning process - based on Scrum with epics, themes and user stories - drove the development schedule and the feature prioritization. By using an agile methodology we were able to change the development scope frequently while retaining a high quality level at reasonable cost.

Using Scrum for developing the core platform was the right decision, especially as it practically enforced test automation, but in future projects we'd put more emphasis on synchronizing the backlogs of the project teams. In fact we developed quite independently from each other (bound to the negotiated APIs) which sometimes blew our integration schedule (i.e. if required functionality was delivered later than anticipated). This lead to delays in the integration of the text processing and subsequently the visualization functions (which to a large extent depend on the topics and categories).

2.2 Organizational Constraints

The budget distribution between the WP3 partners is fixed, which limits the organizational freedom to use these budgets.

Changes to the resource allocation per partner happened and triggered several changes to the DOW.

2.2.1 Organization and Structure

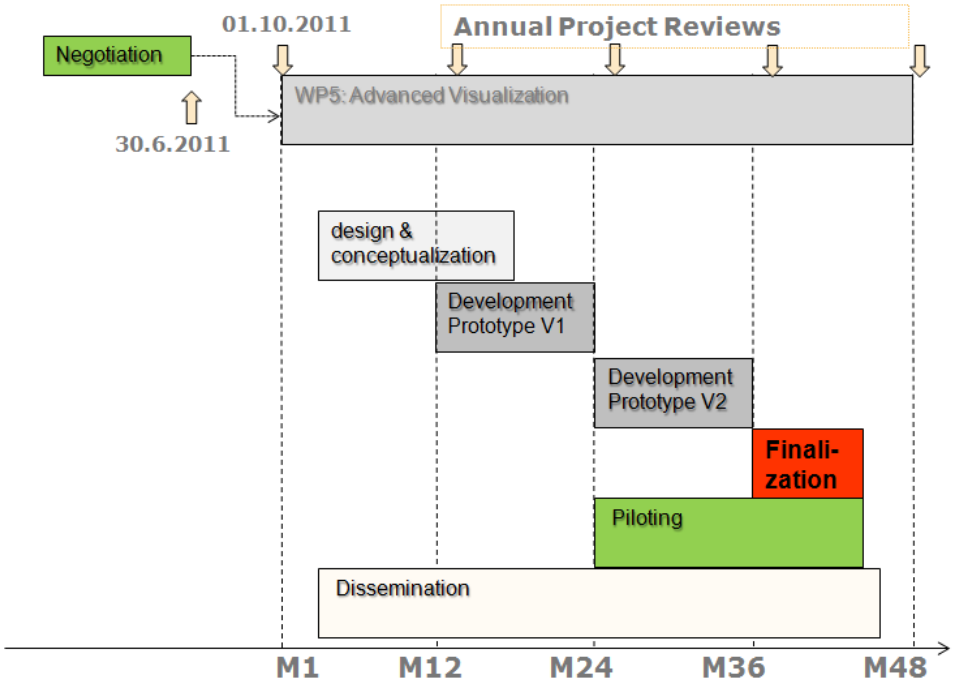
WP3's organizational structure includes resources from several project partners, mainly cellent, Active Solution, Qualysoft and Interfusion for research and engineering tasks.

This has lead to some confusion at the reviews, as it was unclear to the reviewers which participant was responsible for which feature or who executed which task. Although the questions could finally be answered satisfactorily, we didn't anticipate them when designing the work package's organization, which was based on an integrated team that processed items fom the same backlog, instead of an organization that is composed of several teams (one team per participant) with each of them processing their own backlog.

2.2.2 Resources

2.2.2.1 Schedule

The project's top level schedule was as follows and has been granted by the project sponsor:



As the core platform is central to the FUPOL system we decided to put most resources in year one and two. This resulted in a usable core platform from year three on that was used by the other project partners as a foundation and to perform their pilot tests.

However there were significant delays in text processing and (subsequently) visualization that we could only solve in year three. Overall the schedule was good and we still had enough resources in year three and four to fulfill most requests from pilot cities or to cope with the issues that we faced with the semantic data store.

2.2.3 Organizational Standards

2.2.3.1 Development Process

As already mentioned we used SCRUM as the development process, which worked just fine.

2.2.3.2 Quality Standards

The project's software tests were based on the ISTQB standard, which was adequate and worked well.

The decision to automate most tests added much value to the overall progress. However, we automated the core platform's tests only, and not the integration tests to the other modules (i.e. text processing). In a future project we'd try to improve the integration by applying principles from test-driven-development here, as we lost some time in (re-)integrating functionality that was still too buggy to be integrated. This and the change of the algorithm was one of the causes why the availability of the text-processing features was delayed to year three.

2.2.3.3 Development Tools

The following toolsuite was used to develop WP3. All of the tools worked well and we had no major issues.

Tool	Product	Usage	Remarks
Issue tracking system	Atlassian JIRA	• management/tracking of issues (defects) • task management • requirements management (use cases, atomic requirements)	We developed custom types for atomic requirements and use cases, based on snowcards as proposed by IREB/Volere. Furthermore JIRA is

			used for agile planning (JIRA Agile Plugin
Test management system	Tarantula	• test management • test result management	Dropped in year two, as we started to use JIRA for that purpose
Collaboration system (wiki)	Atlassian Confluence	• project communication • requirements elicitation & documentation • technical documentation (pre-official stages) • glossary	We developed several templates for the wiki including a wiki representation of the Volere template. Every work package has its homepage in the wiki.
Continuous integration server	Jenkins	• building of deployables • automated regression testing • generating test coverage metrics • performing continuous integration	
Configuration management system	Maven	• management of the generated deployables	
Document management system	Microsoft Sharepoint	• document management • document versioning	Sharepoint holds the 'official' documentation (some of it is generated as PDFs from pages in the wiki)
Groupware system	Microsoft Exchange	• email (including mailing lists) • adress book • calendar	

2.2.3.3.1 Development and Staging Servers

WP3 used the following stages:

- development stage – the developer's personal PC
- continues integration stage – for building and to perform basic automated tests (a simplified subset of the automated test suite that produces results within a few minutes)
- test stage (2) – an on-premise virtualized system for manual tests and another one for test automation
- demonstration stage – an on-premise virtualized system for the pilot cities and to support exploitation (demo runs and presentations)

During the project additional stages popped up from time to time for experiments (i.e. for the benchmarks) and in year four we added a production stage on SAP HANA Cloud Platform, which is the first cloud based system.

One of the challenges we had was that the maintenance of those server stages was more time-consuming than anticipated for the developers, as the number of technologies and products involved was quite high for every stage (i.e. Postgres with PostGIS extension, Virtuoso, several Tomcat instances, test automation servers, ...) and in order to generate reliable results (reproducable results) the system had to be reset to a defined state before any tests could be performed. While this is true with any software development project we had to perform the same procedure with other teams as well (i.e. text processing) and the distributed data stores didn't make this easier.

In a future project we would put a stronger emphasis on synchronizing the backlogs and the release schedule, including mandatory tests before the integration can be started.

2.2.3.4 Configuration and Version Management

We used GIT (SVN in the earlier stages) for source code management and this worked quite well.

However the work packages maintained their own code base individually and there was no centralized versioning mechanism in effect. This was largely caused by the fact that the work packages used different technologies and programming languages for developing their modules (which is perfectly ok with SOA), and of course by the background that the project partners brought with them (i.e. Xerox' text processing library or Fraunhofer's SEMAVIS). For example the core platform is developed in Java, the text processing module in Python and SEMAVIS in Adobe Flash (Action Script).

An integrated build over all modules would not've added much value to the project.

2.2.3.5 Test Tools and Processes

We used a combination of JUnit (test driver), Selenium (http tests and scripting), JIRA (test definition, test result documentation), Solar (code quality server) and Jenkins (continuous integration server), together with some plugins, for planning, documenting, programming, automating and executing the core platform's tests.

These tools worked quite well, but as already mentioned the maintenance of the test environment was cumbersome.

2.2.3.6 Acceptance and Release Processes

New releases were shipped after each sprint (every two weeks) during the regular development time (mostly year one and two) and upon request or when new features were available during year three and four.

Test automation added significant value to our capability to ship at a fast pace. For example bugs could be fixed and a new release usually be deployed within short time.

2.2.4 Legal Factors

2.2.4.1 Data Privacy and Security

Privacy is a major issue in all systems that process personal data, even more in all domains related to policy making.

Restrictions originate from...

- data protection laws
- copyright laws
- other laws
- terms and conditions of the content providers
- cultural and political expectations

In order to build up trust and to achieve a high degree of acceptance the FUPOL platform had to meet these expectations.

During the project the political and cultural view on data protection (not so much the legal one, which was and is very strict) changed significantly, mostly triggered by Mr. Snowden's disclosure of the NSA leaks. As already mentioned this lead to changes in the data sharing policies of some social media sites and prevented us from getting access to Chinese content, severely hampering pilot operations in Yantai.

Overall, we feel that we couldn't meet the privacy requirements satisfactory as they're highly controversial.

- commercial social media monitoring tools seem to ignore legal restrictions (privacy, data protection, copyright, ancilliary copyright) until they're sued and fly either under the radar or rely on their commercial or market power.
- military or intelligence media monitoring – according to the NSA leaks - ignores all related laws and applies techniques (signal intelligence, large scale

wire tapping, …) that clearly aim at de-anonymizing users. They use every technology that's available to solve their interests.

- municipalities on the other hand are of course obliged to adhere to the law and thus many of them can't make use of social media monitoring except under specific circumstances (that are usually not met in their daily business) – for example in Austria even the police is not allowed to access Facebook (in fact they block Facebook access in the interior ministry) for monitoring without a specific, documented and accepted cause. However, some cities do make use of social media monitoring, but we're not sure if this is based on careful legal analysis or on deliberately avoiding noticing the legal situation.

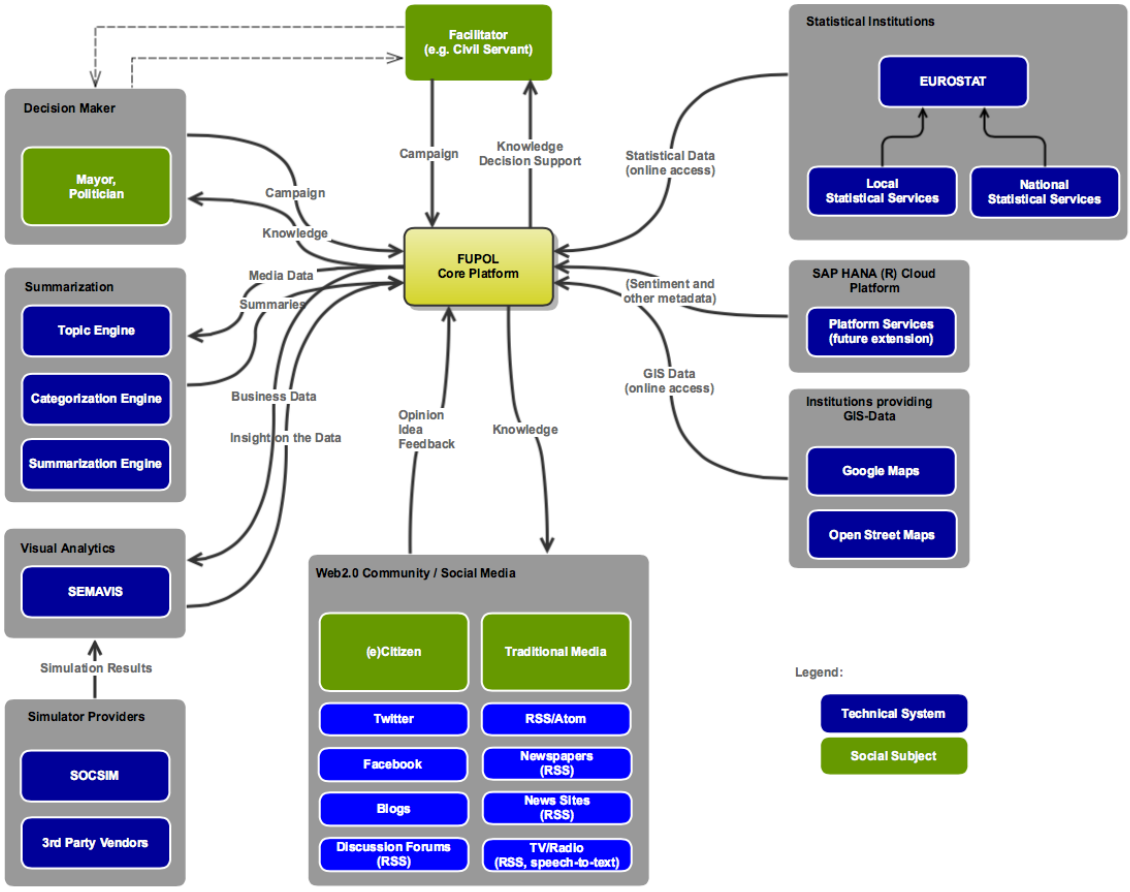
So there's a significant mismatch in what's possible with today's technology, what's allowed (or explicitly forbidden) and what's finally done, especially for military or commercial applications. Even if the sensitive data protection and privacy issues are ignored there's still the more commercial limitations that copyright or ancilliary copyright laws impose. For example Germany's ancilliary copyright prohibits the viewing of newspaper snippets without prior obtaining a license from the copyright proprietor.

The solution for this problem is out of the scope of our project, but we tried to avoid the most obvious and sensitive issues of linking content to a person (de-anonymizing an account), profiling users (i.e. by extracting features like gender from language analysis) or analyzing their spatial movement (though we have the data for doing that).

3 System Scope and Context

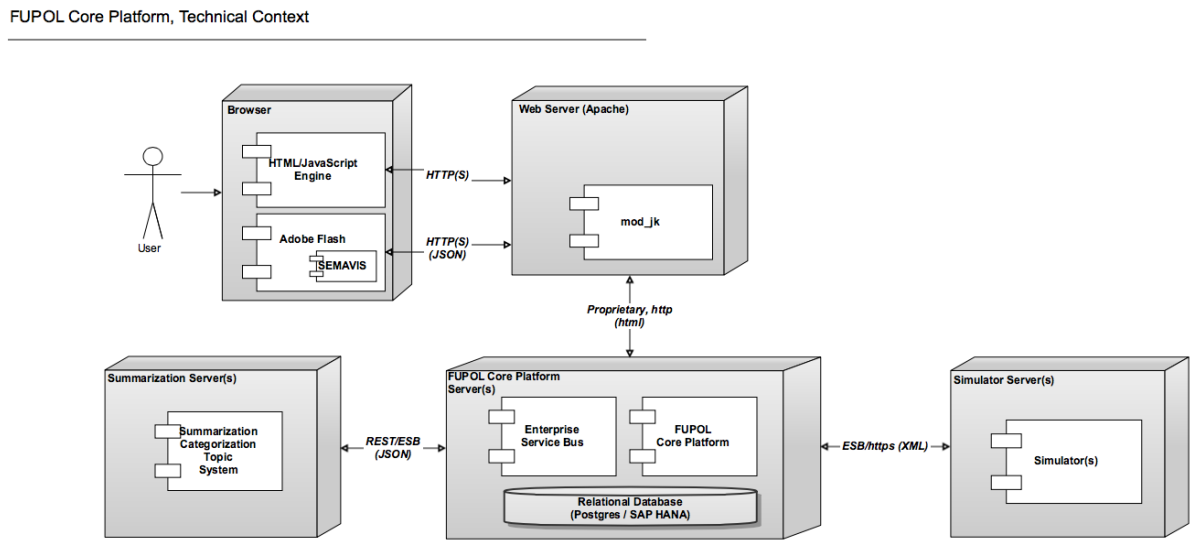
3.1 Business Context

An up-to-date description of the business context can be found in D3.11 (final core platform). The following diagram was copied from that deliverable without further explanations of the details. Please refer to D3.11 for additional explanations.



3.2 Technical- or Infrastructure Context

FUPOL is a distributed system and a such uses several other (internal) modules for delegating specific tasks to them. The following diagram shows the main components of the FUPOL system as seen by the core platform:



The data flows are mostly the same as proposed in D3.2, with the following changes:

- as we phased out the RDF store we had to replace SparQL by REST and RDF by JSON for communicating with the summarization server and the visualization (SEMAVIS). The change was trivial, but of course it delayed the availability of the visualization.
- the relational database that the core platform uses can either be Postgres (traditional on-premise installation) or SAP HANA (on-premise private cloud or in the public cloud)
- the data flow between the core platform and the simulator(s) was reduced to authentication, as the core platform doesn't use/store the simulation results (the simulators provide their result data directly to a client-embedded version of SEMAVIS and store it locally).

More information on the data flows can be found in D3.2 (considering the changes above).

4 Solution Ideas and Strategy

4.1 Architectural Strategy

The architecture follows a service oriented approach.

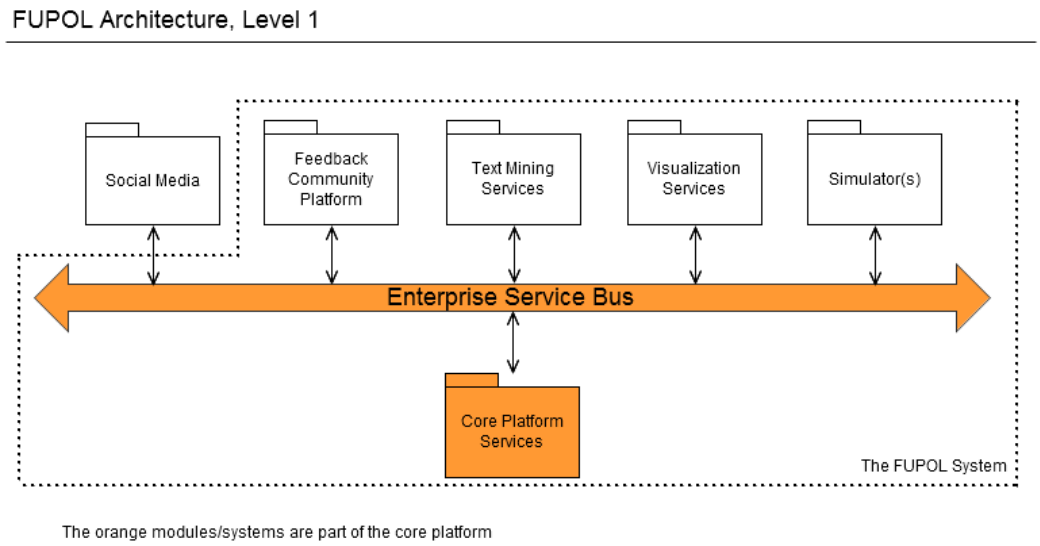
The reasons for this are:

- the core platform is the foundation for a homogenous product built from modules that are implemented in various technologies (Java, Flex, Python, C++, ...)
- as FUPOL is a research project we had frequent changes (i.e. text processing changed its data structures, algorithms and the related API several times) in scope and so we had to go for a high level in architectural flexibility. This rules out tight coupling.
- the requirement of multi-client support (WP3-33, see D3.1) induces strict security requirements that are easier to enforce if there's some central communication middleware
- we must be able to add/remove modules with relative ease

5 Building Block View

5.1 Level 1

The actual architecture is based on an a service oriented architecture using an enterprise service bus (ESB). Note that most connections between the modules are (logically) point-to-point, but technically all communication is done through the ESB.



A description of the modules can be found in D3.2. Note that the feedback community platform was only envisaged, designed and in an experimental stage, but didn't make it into the final product.

As already mentioned the SOA had impact on the performance and scalability of the system (see the benchmark test results).

In a future project we would design a tighter coupling between the chatty components (core platform to text mining and vice versa) and the visualization service would get an abstraction layer to reduce the traffic between client and server (currently all visualized data has to be transferred between the core platform and SEMAVIS, which limits its use to detailed analysis and leaves the statistical overview to the simpler visualizations of the core platform or to the currently being under development self-service BI solution).

D3.11 provides a data flow diagram.

5.2 Level 2

The level 2 of the core platform's architecture is very similiar to the one outlined in D3.2 and we only describe the differences here.

5.2.1 Web Clients

The FUPOL Core Platform provides two different clients:

- FUPOL Console
- FUPOL Administration Console

Both consoles are accessible using web browsers.

FUPOL Console

The FUPOL Console provides user access to the core services. This is the application that the internal users (facilitator, decision maker, domain expert, ...) utilize. Furthermore it's able to render external content (like the opinion maps) that can be accessed from third party systems (i.e. the city's blog).

Rendering content that is embedded in external systems (i.e. blogs or Facebook) was a good decision, as it allowed us to get a higher reach for the citizens. For example Pegeia used the opinion maps directly in Facebook and was able to generate significantly better response than the pilot cities that placed their opionion maps just on web pages.

FUPOL Administration Console

The FUPOL Administration Console or (FUPOL SysOp Console) is for internal use by the FUPOL service provider's staff. It's main purpose is the management of the clients (where a client is a pilot city).

The FUPOL Administration Console can only be accessed by users with the role 'system operator' and it didn't evolve significantly from the initial release.

5.2.2 Access Management (Black Box Description)

Access management relates to enabling users to access the FUPOL Console or revoke those rights from them.

FUPOL uses an internal account management system (see the next chapter) and form based authentication for all internal users. eCitizens access the system by providing their credentials from other systems (Twitter or Facebook) and we use OAuth for authenticating them.

Using OAuth proved to be harder than anticipated because the involved callbacks were limited by the mix of http and https (SSL) when we embedded our content as iframes on (external) web pages. Some browsers (or versions of them) considered this mix as unsafe and showed a warning or refused the call back connections.

5.2.3 Account Management (Black Box Description)

Accounts are managed in the FUPOL Console (except for the system operators and administrators).

There's currently no enterprise integration (i.e. Active Directory) available, but we might inherit this feature from HANA.

5.2.4 Campaign Management (Black Box Description)

Campaign management was implemented as planned.

5.2.5 Client Management (Black Box Description)

Clients are managed in the FUPOL Administration Console. This includes creating/locking them and setting their defaults (i.e. if a client prefers to use Open Street Maps or Google Maps for the base maps).

5.2.6 Crawler (Black Box Description)

The crawler was implemented on a point-to-point basis so that the core platform performs calls to each social media API individually. A scheduler (with manual settings) controls the request frequency.
In a future project we would outsource this function to a social media aggregator (i.e. Datasift) in order to reduce the effort that is required for developing and maintaining those indivudual connections.
A special feature of the crawler is that we implemented a generic web reader using boilerplate code elimination. This is used for extracting newspaper articles that are linked from RSS streams. Boilerplate code elimination is more CPU-consuming than interpreting a tweet, which is quantified in our benchmarks, but usually the number of interpreted pages was much lower than with high volume streaming-data like Twitter's posts.

5.2.7 Data Management (Black Box Description)

Data are values of a qualitative or quantitative variables, belonging to a set of items/themes ("facts of the world"). As an abstract concept data can be viewed as the lowest level of abstraction from which information and then knowledge are derived.

The data management as used by FUPOL had to be changed significantly, mainly caused by the stability issues with the RDF store. However, the final system uses the SAP HANA in-memory database, which is optimized for analysis and BI and a very good foundation for later exploitation. With this move we can add self-service real-time BI solutions to the collected data and provide campaign-individual dashboards and views on the data, something that most commercial media monitoring tools lack.

This will be one of the competitive advantages that we'll exploit to sell FUPOL.

5.2.8 Knowledge Management (Black Box Description)

Knowledge is a familiarity with someone or something, which can include information, facts, descriptions, or skills acquired through experience or education. Briefly said knowledge is what we know.

Knowledge management has undergone several changes throughout the project. In the early stages an internal knowledge base, fed with external data and data that was generated using FUPOL was envisaged ("city knowledgebase"), but this idea proved to be less relevant for the cities, as there's plenty of public data available for their needs.

So we focused on providing (quantified) facts by adding a statistical data layer on top of our RDF store (Virtuoso) and provided functions for importing statistical data from Eurostat, while WP5 enhanced SEMAVIS with widgets for viewing numerical data (SEMAVIS is designed as a semantic data browser). Importing data from Eurostat worked well, but unfortunately their semantic data format for statistical data (SDMX) was not adopted by most cities, though the national statistical organizations started using it. In fact, we were unable to get relevant statistical data from cities in SDMX format.

So the final solution is that SEMAVIS is directly accessing public data from Eurostat and others, as storing this public external data in the core platform is quite pointless.

5.2.9 Operational Support (Black Box Description)

Operational support includes access to the log files.

5.2.10 Social Media Management (Black Box Description)

The term *social media* refers to the use of web-based and mobile technologies to turn communication into an interactive dialogue.

A *social media window* is both a concept and an important domain object in FUPOL. As a concept the term refers to the idea that a facilitator is interested in accessing several social media sites in a convenient way ("single window to social media"). The domain object is a container for social media access and the associated results (content).

Social media management was implemented for the following media:

- Facebook (public pages; support for private pages was dropped by Facebook in 2015)
- Twitter
- RSS (mostly used for importing newspaper or forum content)
- Sina Weibo (worked technically, but access was revoked by the service's operator before we could start any pilot tests)
- Opinion Maps
- Questionnaires

Opinion maps and questionnaires are tools that we added upon request from the pilot cities, but their data is processed in the same way as the other social media data.

As an enhancement we added support for EMM's speech-to-text services which allowed us to finally support social media, newspapers and broadcasters with the system, adding the selling point of 24x7 media coverage to the system.

5.3 Level 3

More details on level three of the design can be found in deliverable D3.2. Again, we just comment on the relevant differences between D3.2 and the final version.

5.3.1 Campaign Management (White Box Description)

As already mentioned we added questionnaires to the campaign's tools, as a simple example for a e-participation tool requested by cities.
FUPOL's questionnaires (and the opinion maps) have the following advantages over most existing commercial solutions:

- they can be embedded on external sites (as iframes), including social media sites
- in case of authenticated access users can apply their Facebook/Twitter credentials and don't have to create yet another account
- the tools that FUPOL provides can be used on questionnaires and opinion maps – the data store is integrated

Technically both tools are web pages that are generated by the core platform, which provides their URL to the user. The URL is then used to insert the FUPOL-generated content into another site.

Both tools support OAuth 2.0 with Twitter and Facebook. The planned support for Sina Weibo's OAuth servers had to be dropped for reasons already mentioned.

5.3.2 Data Management (White Box Description)

FUPOL's data management has changed as the project progressed. While the initial idea was to utilize semantic web technology for accessing external data and storing it internally (RDF store), the available products had serious issues with stability or performance.

We used...

- Apache JENA SDB/TDB (native or backed by Postgres; mostly for development in the earlier stages)
- Virtuoso for test and pilot operations

While Apache JENA's stores were too slow to fit (as we anticipated), we were unable to operate Virtuoso in a way that fulfilled even our lowest requirements for platform stability.

As outlined in D3.11, the issues can be summarized as:

- availability decreased over time – the response time increased significantly under heavy load and the system was unable to restore normal operations once the load was reduced, so we had to restart it almost daily
- under certain circumstances that we were unable to understand fully the stored data became inconsistent, which was detected and reported by the datastore, leading to labour-intensive clean-up operations
- furthermore we faced issues with Virtuoso's JDBC driver that we had to fix manually

The observed effects increased under heavy load, especially when concurrently inserting RDS triples/quads to the same resource. Coping with these issues consumed too many resources, so we had to drop Virtuoso in year three and substituted the RDF store with a simple relational database (Postgres), preserving the overall data structure, which is based on FOAF, SIOC and DC. The Postgres based version was used for the pilot tests in year four.

However, during year four we took the opportunity to migrate the core platform to a more advanced database system (SAP HANA), which solved several project issues for us and which provides a solid basis for future development and commercialization, and – given the possibilities that HANA provides – another selling point:

- By using SAP HANA, which is a platform-as-a-service product, we are able to operate the core platform as a cloud based service, even to a very large scale (i.e. the NSA uses HANA)
- FUPOL can be offered and sold in SAP's cloud store, which provides an additional sales channel and supports exploitation
- By operating the core platform on the HANA cloud platform we make use of a very advanced in-memory database which is optimized for analytics (though our benchmark tests showed that the overall speed of the current core platform's data processing is comparable to the less advanced Postgres version of FUPOL). However, using the column-oriented store adds the ability to perform real-time analytics to the collected data. This is something that can't be done with the Postgres-based version, as the analytics tools usually work on a copy of the root dataset that's cloned at discrete points in time (i.e. daily).
- For future exploitation HANA – being a platform-as-a-service solution – provides platform services that might be used to extend FUPOL's functionality to areas that were not covered by the project (i.e. sentiment analysis) at low cost (as compared to develop it by the consortium)

To get a better understanding of the potential that this move brought into the FUPOL system, a technical introduction to SAP HANA is provided.

According to SAP, HANA is based around „a common database approach for OLTP and OLAP using an in-memory columnar database". The business driver behind that is that data management for today's analytics applications requires real-time data processing, while most existing solutions work on an isolated copy of some database. So HANA implements the move from batch-based analytics (which is decoupled regarding to time) to real-time analytics.

The following table illustrates this paradigm change:

Aspect	Traditional Analytics	Real-time analytics
Data stores	1. transaction store (=root data) 2. analytics store (copy of 1) 3. acceleration store (copy of 2) Data is moved from 1->2 at discrete points in time (i.e. hourly) and from 2->3 upon request	1. transaction store (=all data) There's only one source for the data and the in-memory data store is fast enough to fulfill the requirements for storing, analyzing and caching the data
Latency	High Depending on the data store (1, 2 or 3)	Very low (random access)
Cost per byte	Low	High (but decreasing and already low enough for many applications)

Besides the in-memory database HANA provides additional application services and most of them are directly running in the in-memory database platform, making the overall system very fast.

These services include (those that we deem to be relevant for extending FUPOL are in *italics* letters):

- *operational analytics* (analyzing existing data, comparable to the core platform's campaign dashboard)
- *predictive analytics* (anticipating future changes in the data, i.e. events)
- machine learning (learning i.e. decision trees from the data)
- prescriptive analytics (scenario-based predictions)
- *text processing* (supporting many languages and with features that are out of the scope of our project)
- *sentiment analysis* (supporting many languages)
- *data streaming*
- planning
- transations
- *geospatial analytics and GIS*

FUPOL currently only uses the HANA database and the platform's Java runtime (including the Tomcat server), but we might extend the system later to make use of some of these services (i.e. sentiment analysis). Note that by integrating the services from the datastore we could overcome some of the limitations that are imposed by our SOA architecture, especially in text processing. Migrating the HTS functions from Python to Java and executing them closer to the database should give the system a huge performance boost (note that most of our database traffic is related to HTS – see the benchmarks in D3.11 for details) and thus open the system to near real-time topic extraction and categorization.

So the overall idea of HANA is that it's not only a database, but a platform and the application layer (which is for most part the core platform, the visualizations and the simulators) operates on a rich set of database-bound services, delegating the database-intensive work to optimized off-the-shelve services.

The defining features of the database (that we already use) are:

- data is kept in-memory as long as possible and the available DRAM is „huge" (there's a transparent paging mechanism available that moves data to slower storage if it doesn't fit into the DRAM). Thus no aggregates need to be precalculated – all data models are calculated on-the-fly
- the data is organized in columns instead of rows, thus eliminating the need for indices. Columnar data access is optimal for most analytics applications, i.e. aggregations of column values and well suited for our data semantics (i.e. tagging).
- the columnar data store is auto-compressing (data is only stored once; copies refer to the single instance).
- data access can be parallelized (over CPUs, processes and machines) and is thus much faster for processing big data

The following table shows the progression of the FUPOL platform over time, starting with the Postgres-based version of year three, moving on to the HANA based version

of year four (which is the final version) and a potential future version (which might be developed after the project ends):

Architectural Layer	**Classic FUPOL** **(Postgres)**	**Current FUPOL** **(Hana Db)**	**Future FUPOL** **(Hana Db + Services)**
User interface	Wicket + browser SEMAVIS, simulators	Wicket + browser SEMAVIS, simulators QlikView	Wicket + browser SEMAVIS, simulators QlikView or Lumira
Service Logic	core platform	core platform	core platform
Data-centric logic	core platform, text processing, simulators	core platform, text processing, simulators	core platform, text processing, simulators, Hana services
Database	Postgres	Hana HCP	Hana HCP

Besides the database FUPOL uses HANA's Tomcat application server, so the whole core platform is now available in the cloud.

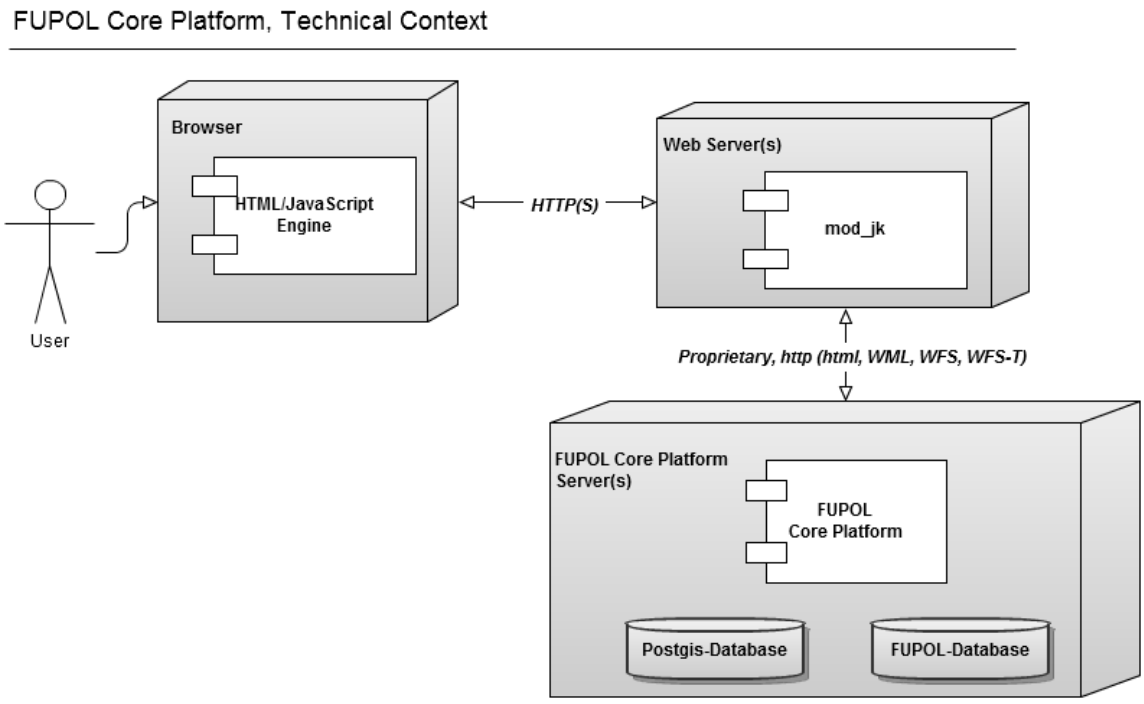
6 Runtime View

The runtime view provides some insights on how the FUPOL Core Platform works dynamically. A comprehensive description can be found in D3.2.

7 Deployment View

Technical- or Infrastructure Context

FUPOL is a distributed system and a such uses several other (internal) modules for delegating specific tasks to them. The following diagram shows the main components of the FUPOL system as seen by the core platform:



Note that the Postgis-extension to the FUPOL database, which we used in year one and two, was removed from the system, as it was no longer necessary.

Both versions (Postgres and HANA) use Tomcat as a web/application server, with the Postgres system on Tomcat 8 and the HANA system on Tomcat 7 (as Tomcat 8 is not yet available on HANA).

Physical Servers and Development/Test/Demo Hardware

The Postgres-version is currently hosted on our virtualized on-premise server (located at cellent's office in Vienna). All stages (test, demonstration, …) are hosted here.

The HANA version is hosted in SAP's datacenters, which are distributed around the globe:

Note that SAP offers a service that allows their cloud customers to limit the geographical regions that their data is moved to. For example a German city might be obliged that their data doesn't leave Germany or the EU. We were asked for such a service by potential customers several times.

The hardware that runs the Postgres version of FUPOL is now outdated, but still strong enough to perform comparably to out (free) HANA instance in our benchmarks:

The physical server stage includes the following hardware...

- 2x Server IBM System x3550
  CPU: Intel Xeon E5659 2,64GHZ

RAM: 65GB

- 1x IMB Storage TS2900 Tape Autoloader
- 1x IMB DS3500
  12x 800 GB SAS harddisks (RAID)

- 2x SmartUPS 3000 XL (USV)
- 3x Cisco Catalys 2950 (Switch)

...and the following software components:

- VMWare Essentials
- Veeam Backup & Replication

8 Recurring or Generic Structures and Patterns

Generic UI Components

All user interface components are based on the light-weight component-based web application framework Apache Wicket 6.0 (http://wicket.apache.org/, [Igo11], [Joc12], [Mic09], [Ola09], [Ken10], [Ken12]) that we customized to fit our needs.

For reuse we developed many components as part of the FUPOL framework, which was surprisingly difficult, especially for new team members.

9 Technical Concepts and Architectural Aspects

Deliverable D3.2 covers some design topics like inter-module communications, coupling, physical distribution of the system's nodes, exception and error handling, logging, configurability, multi-threading (especially in the GUI), internationalization (we developed localized versions for English, German and Slowak).

Please refer to D3.2 for further details.

10 References and Bibliography

Continous Integration

[Sim10] Wiest, Simon. Continuous Integration mit Hudson. 2010.

Geomatics

[Ant12] Perez, Antonio Santiago. Openlayers Cookbook. 2012.

[Nor05] Lange, Norbert de. Geoinformatik in Theorie und Praxis. 2005.

[Nor051] Bartelme, Norbert. Geoinformatik: Modelle, Strukturen, Funktionen. 2005.

[Dav10] David J. Maguire, Michael F. Goodchild, Paul A. Longley. Geographic Information Systems & Science. 2010.

[Eri11] Hazzard, Erik. Openlayers 2.10 Beginner's Guide. 2011.

Hibernate

[Ric08] Richard Oates, Thomas Langer, Stefan Wille, Torsten Lueckow, Gerald Bachlmayr. Spring & Hibernate: Eine praxisbezogene Einführung. 2008.

Maven

[Mar09] Spiller, Martin. Maven 2: Konfigurationsmanagement mit Java. 2009.

Mule

[Joh10] John D'Emic, David Dossot. Mule in Action. 2010.

PostgreSQL

[Leo12] Leo Hsu, Regina O. Obe. PostGIS in Action. 2012.

Scrum

[Bor11] Gloger, Boris. Scrum: Produkte zuverlässig und schnell entwickeln. 2011.

[Rom09] Pichler, Roman. Scrum: Agiles Projektmanagement erfolgreich einsetzen. 2009.

Social Media

[Mic12] Kamleitner, Michael. Facebook-Programmierung. Entwicklung von Social Apps & Websites. 2012.

Software Architecture

[Dir05] Dirk Slama, Karl Banke, Dirk Krafzig. Enterprise SOA: Service-Oriented Architecture Best Practices. 2005.

Test Automatisation

[Tho11] Thomas Bucsics, Manfred Baumgartner, Richard Seidl. Basiswissen Testautomatisierung: Konzepte, Methoden und Techniken. 2011.

UML

[Iva05] Ivar Jacobson, James Rumbaugh, Grady Booch. The Unified Modeling Language User Guide. 2005.

Java

[Cra12] Walls, Craig. Spring im Einsatz. 2012.

[Gar08] Mak, Gary. Spring Recipes: A Problem-Solution Approach. 2008.

[Jos08] Bloch, Joshua. Effective Java (Second Edition): A Programming Language Guide. 2008.

JavaScript

[Kar12] Karl Swedberg, Jonathan Chaffer. JQuery lernen und einsetzen. 2012.

[Oli12] Ochs, Oliver. JavaScript für Enterprise-Entwickler. 2012.

[Sto11] Stefanov, Stoyan. JavaScript Patterns. 2011.

Web

[Avi09] Kaushik, Avinash. Web Analytics 2.0: The Art of Online Accountability and Science of Customer Centricity. 2009.

[Pas08] Pascal Hitzler, Markus Krötzsch, Sebastian Rudolph. Semantic Web. Grundlagen. 2008.

Wicket

[Igo11] Vaynberg, Igor. Apache Wicket Cookbook. 2011.

[Joc12] Mader, Jochen. Wicket: Komponentenbasiert und objektorientiert - das alternative Java-Webframework. 2012.

[Mic09] Mosmann, Michael. Praxisbuch Wicket: Professionelle Web-2.0-Anwendungen entwickeln. 2009.

[Ola09] Olaf Siefart, Carl-Eric Menzel, Roland Förther. Wicket: Komponentenbasierte Webanwendungen in Java. 2009.

[Ken10] Tong, Kent Ka Iok. Developing Web Services with Apache Axis. 2010.

[Ken12] Tong, Kent Ka Iok. Enjoying Web Development with Wicket. 2012.