MOVING

Deliverable 4.1: Definition of platform architecture and software development configuration

Sebastian Gottfried, Paul Grunewald/TUD
Alexandros Pournaras, Chrysa Collyda/CERTH
Angela Fessl, Peter Hasitschka/KNOW
Aitor Apaolaza/UMAN
Ahmed Saleh, Till Blume, Ansgar Scherp/ZBW

31/01/2017

Work Package 4: Iterative MOVING platform development

TrainInG towards a society of data-saVvy inforMation prOfessionals to enable open leadership INnovation

Horizon 2020 - INSO-4-2015
Research and Innovation Programme
Grant Agreement Number 693092

Dissemination level	PU
Contractual date of delivery	31/01/2017
Actual date of delivery	31/01/2017
Deliverable number	4.1
Deliverable name	Definition of platform architecture and software development configuration
File	`moving_d4.1-v1.0.pdf`
Nature	Report
Status & version	Final v1.0
Number of pages	28
WP contributing to the deliverable	4
Task responsible	TUD
Other contributors	CERTH, KNOW, UMAN, ZBW
Author(s)	Sebastian Gottfried, Paul Grunewald/TUD Alexandros Pournaras, Chrysa Collyda/CERTH Angela Fessl, Peter Hasitschka/KNOW Aitor Apaolaza/UMAN Ahmed Saleh, Till Blume, Ansgar Scherp/ZBW
Quality Assessors	Peter Mutschke/GESIS
EC Project Officer	Hinano SPREAFICO
Keywords	Architecture, supporting software, quality assurance

Table of contents

List of Figures

List of Tables

Executive summary

This deliverable contains the technical documentation of the architecture of the MOVING platform and the feasibility studies for the projects' use cases. The architecture of the MOVING platform is based on the requirements as described in the DoW and will be refined after the delivery of "D1.1: User requirements and specifications of the use cases", which is on M12 of the project. The MOVING platform provides an integrated working and training environment for data professionals. It is based on TUDresdens' eScience platform[1] and fulfills the requirements for developing an advanced search solution with novel visualisation techniques and curricula with relevant educational material for the training of data professionals. The deliverable presents the architecture of the MOVING platform that is capable of implementing the aforementioned features and validating them into the projects' use cases.

Finally, the deliverable includes a description of the supporting software for the development processes and the quality assurance measures for the project.

[1] https://github.com/tud-mit-plone/escience, last accessed at 23.01.2017

Abbreviations

Abbreviation	Explanation
DoW	Description of Work
SCM	Source code managment
SQL	Structured Query Language
ATS	Adaptive training support
JSON	JavaScript Object Notation
NoSQL	Non Structured Query Language
DOM	Document Object Model
REST	Representational state transfer
cMOOC	*connectivist* massive open online course
MVC	Model–view–controller
QA	Quality assurance

1 Introduction

1.1 History of the document

Date	Version
14.12.2016	v0.1: Initial TOC draft
02.01.2017	v0.2: Second TOC draft
03.01.2017	v0.3: Draft with requirements
16.01.2017	v0.4: Draft
16.01.2017	v0.5: Draft
23.01.2017	v0.6: Draft
24.01.2017	v0.7: Draft
30.01.2017	v0.8: Draft
30.01.2017	v1.0: Final version

1.2 Purpose of the document

This documents evaluates options for the architecture of the MOVING platform based on the requirements in the Description of Work (DoW) (see Section 2). It provides a technical description of the chosen architecture (see Section 3). To validate this architecture we check if it is possible to complete the two use cases with the proposed platform (see Section 4). Finally, we describe the supporting software configuration (see Section 5) and the quality assurance (QA) measures (see Section 6) in the MOVING project.

2 Requirements

In this section we list the initial set of the requirements according to the DoW. The requirements will be refined accordingly as soon as we have the final input from D1.1 which contains the results of the requirement analysis and will be delivered in M12 of the project. The following list with the initial requirements is used in order to assess the different architecture options of the MOVING platform:

eScience platform The MOVING platform should be based on the already existing research platform, the eScience platform[2]. It is a working environment for the advanced digital initiation, organisation, implementation and documentation of projects in the academic field. This allows us to reuse its features for the integrated working and training environment we intend to create with MOVING. To this end, the existing platform should be extended to meet the requirements derived from the use use cases defined by MOVING.

Curricula The existing eScience platform should be extended by tools for modelling, representing and the "execution" of curricula (from WP2). We will produce curricula for the training of data-savvy information professionals which will be implemented within the platform.

Adaptive training support The adaptive training support (ATS) helps the MOVING platform users to increase users' awareness of their training progress and facilitate the transfer of newly learned theoretical knowledge to work-related situations (from WP2). This integrates the continuous, professional training with real work life.

Search There should be an advanced search solution with novel data visualisation techniques (from WP3). Scientific publication from open archives, public web pages and social media should be crawled and indexed. In this process, semantic concepts should be extracted and stored for later retrieval. The learned concepts should be made available to the user with interactive data visualisations, e. g. with concept graphs.

User management integration The platform should have the capability to integrate into the existing user management facilities of institutional IT environments.

[2] https://github.com/tud-mit-plone/escience, last accessed at 23.01.2017

User tracking The platform should be instrumented with a logging mechanism that connects to UMAN's log analysis dashboard. The collected user tracking data should be used as the foundation for the prior knowledge assessment in the adaptive training support.

Since these are only the initial requirements, we will try to find an architecture which is flexible enough to also accommodate the upcoming requirements during the projects' progress.

3 Architecture of the MOVING platform

3.1 Overview

In this section we present the architecture for the MOVING platform. Figure 1 gives an overview of the components of the architecture.

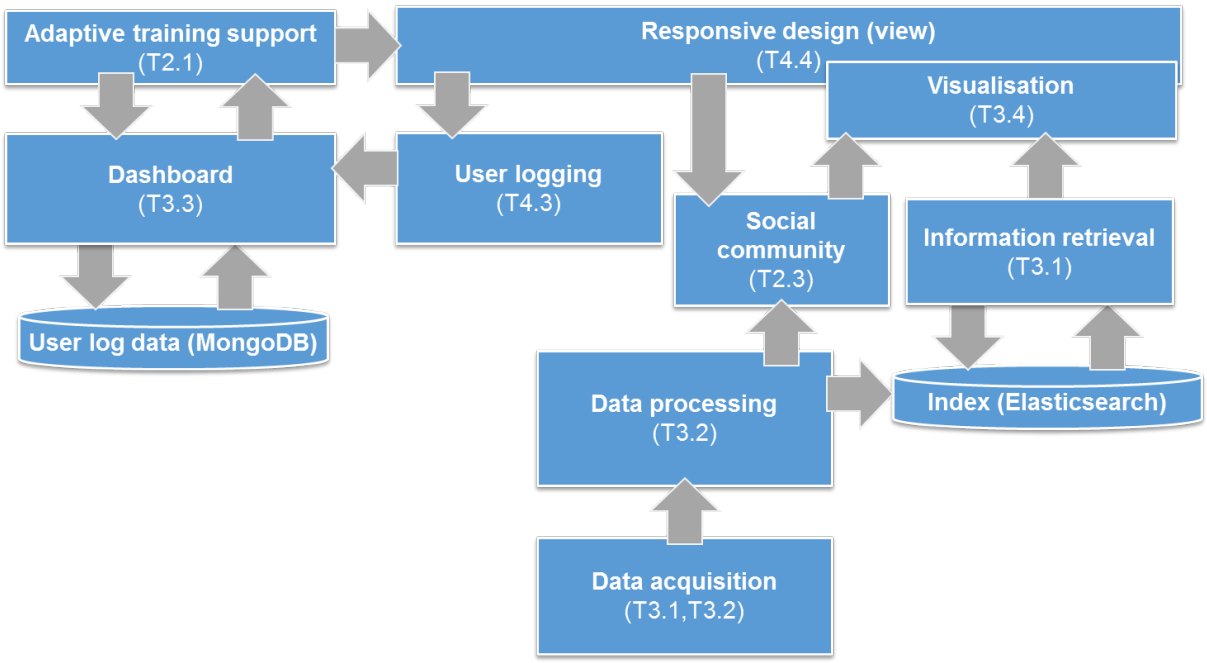

Figure 1: High level overview of the interaction between the different components of the MOVING platform

The architecture comprises of four main components which are described in the following subsections. Since the MOVING platform will be a web-based, we implemented a web application (see Section 3.2) to host the user-visible functions of the platform. To realise the advanced search capabilities for the platform we developed a search engine (see Section 3.3). Which is fed with data from the intended data sources. This is the task of the MOVING crawler (see Section 3.4). Finally, there is a component to track and analyse the users' behaviour on the platform (see Section 3.5). It will be used for the adaptive training support to select appropriate training material depending on the users' usage patterns of the application.

3.2 MOVING web application

3.2.1 Extension of the eScience platform

The MOVING web application will is based on the existing eScience platform. In this section we introduce the eScience platform and explore the options to realise the extensions needed for the MOVING platform.

The eScience platform is a web application which provides an online research tool. Users of the platform can create projects in the platform on which they can collaborate with other users. For each project the user can choose from a set of modules to enable specific functionalities. Currently there are modules providing a project wiki, task management, time tracking, document management, blogs, file storage, polls and message boards. Additionally users can maintain a profile page with their research fields and competencies, that are accessible to other users via search. Finally there is a messaging system which users can use to connect with other researchers.

Technically, the eScience platform is a heavily customised version of the Redmine project management system[3] originally developed during the project *eScience research network Saxony*[4] at the HTWK Leipzig. It has been used to successfully host 25 research projects during the project's lifetime. Since then, the software has been further developed at the Media Center of the TU Dresden and is currently introduced as a research service for the members of the university.

The architecture of the eScience platform is heavily influenced by the web application framework the Redmine developers have chosen, *Ruby on Rails*[5]. This framework aims to make the process to develop a web application as easy as possible. To achieve this goal, it imposes strict conventions on the application architecture and development process. It organises the application programming by the heavy employment of the model-view-controller (MVC) design pattern (Krasner & Pope, 1988):

- A *model* describes an entity type within the application. It is implemented as a Ruby class. The framework has an object-relational mapping system, ActiveRecord[6], which takes care of persisting and loading instances of the model to or from a database system. This process relies for the most part on convention and very little code is required to configure the mapping. The framework can infer the database layout from the model structure. The use of SQL is only required in special situations.
- *Controllers* host the logic of the application. They handle incoming HTTP requests, query or store information from the model layer and select a view to create the response for a request. The framework contains a configurable router to map URLs and HTTP methods to method calls.
- A *view* is a special kind of HTML file with embedded Ruby code to realise the dynamic parts. One view can embed other views, which only render a specific part of the result page. This allows reusable view components which can be shared by multiple views. The framework can also generate XML or JSON responses from model instances without the need of a dedicated view. This is useful for the creation of web APIs.

Ruby on Rails comes with an extensive suite of tooling to guide every aspect of the development process. The scaffolding for new models, controllers or views can be generated with database migrations and out-of-the-box structures for test cases. It also includes a built-in web server to run the application in development mode and a test runner with coverage analysis.

Redmine provides its own set of abstractions on top of Ruby on Rails to ease the development of project management-related functionalities:

- While Ruby on Rails only ships with a basic framework for authenticated access to the web application Redmine has a full-fledged role-based authorisation system. One can define permissions bound to controller methods restricting the access to these methods. These permissions are given to roles. Users can be assigned roles either application-wide or within the scope of a project.
- There is a menu system to define the contents of the navigation menus declaratively. It is used to construct the application navigation by defining several menus for the application's navigation axes. Menu items are bound to a controller method and can have a label, dynamic parameters and a condition to show an item only in specific cases.
- Redmine provides several extensions for the model layer. Model classes can be made watchable (users can choose to watch them and are then notified about changes), searchable (model instances show up in the search) or can be an activity (each project has an activity feed with events concerning the project, e. g. new tasks or wiki edits).

Redmine can be extended with plugins. A plugin can provide additional models, controllers or views and can hook into the application by using the aforementioned abstractions. They provide a clean way to add new functions to Redmine without interfering with the application's core.

Changing the behaviour of the application with a plugin is less structured. There is no formal way to override or extend existing application artefacts. The usual way is to patch existing classes with new method implementations (*monkey patching*) to achieve the desired behaviour.[7]

[3] https://www.redmine.org/, last accessed at 12.01.2017
[4] https://www.escience-sachsen.de/, last accessed at 12.01.2017
[5] http://rubyonrails.org/, last accessed at 12.01.2017
[6] https://github.com/rails/rails/tree/master/activerecord, last accessed at 12.01.17
[7] The distinct feature of Ruby that class definitions are *open* helps greatly at monkey patching. If class is defined twice the latter definition will extend or amend the original class. Changes affect both new and existing instances of the class.

The DoW proposes to implement the new functions for the MOVING platform as plugins for eScience platform. This approach would allow independent development of the MOVING functionality from the core. The main advantage would be the possibility to easily integrate new Redmine releases into the project for new functions and bug fixes, including security patches.

The original development team of the eScience platform has started with a set of plugins for Redmine to realise the platform. Later, the team also made considerable changes to Redmine itself. In this phase they also have introduced severe dependencies from the core to the eScience plugins. This makes it impossible to run the Redmine core of the eScience application without its plugins.

We evaluated if it possibility to disentangle these dependencies, but the effort was deemed to high due to the number and complexity of the performed changes. Instead we have chosen the opposite strategy and merged the code of the plugins into the core. The effect was a smaller code base with less complexity and without the mentioned cyclic dependencies.

Because the development of eScience and Redmine has diverged upstream updates cannot be merged anymore to the eScience code base without further changes. This is especially a problem if upstream releases contain security fixes. Therefore we monitor the security advisories[8] for Redmine closely and apply the fixes to the eScience codebase in case they apply.

In this light there is little to gain by implementing the MOVING platform extensions with plugins and we would limit ourselves to the capabilities of the plugin API which certainly does not cover all needs of the proposed changes in the requirements. In particular, it lacks an API to extend the search with a new engine, which is required for the advanced search capabilities. So it can only achieved by a plugin with monkey patches. Since the Redmine search cross-cuts large parts of the application, it would require a large number of patches leading to a potentially brittle application architecture which risks a decrease in overall software quality.

Therefore we have decided to implement the MOVING platform directly in the eScience code base.

3.2.2 General design decisions

We will follow the MVC pattern to introduce new functions that fulfil the requirements. To this end, we will add new models or extend existing ones to model the application entities like training data, search results or curricula and implement the logic on them in a set of new controllers with corresponding views to render the HTML code for the user interface (see 3.2.3 and 3.3.3 for details about the architecture of the user interface). Finally, we will modify existing components when necessary to implement the requirements. Since the exact requirements are not known yet, we provide the following general design decisions for the architecture:

Curricula We will utilise the already existing blog feature of the eScience platform for the creation and consumption of content for curricula. The feature may need to be extended to meet all the requirements, but should provide a solid base for it. Blog posts already support rich content allowing the integration of multimedia content or social networks and allow feedback via a commenting system. These are the basic requirements to successfully host connectivist massive open online courses (cMOOC), which we want to achieve in the MOVING platform.

Adaptive training support For the development of the ATS, we will use a typical client-server architecture utilising well established representational state transfer (REST) API calls. The ATS will use data captured in the user models and from the user interaction tracking (see Section 3.5). In addition, the ATS will use an ATS database to store and retrieve relevant ATS information. Out of this data, relevant information for the ATS will be computed and will be sent in JSON form to the client. The client will then use the available data to render the ATS widget (see Section 3.5.3)

The server side will be implemented as a set of models capable of representing training data and a controller operating on the new models to allow the client querying training data from the server.

Search For the advanced search solution we have deployed a custom Elasticsearch-based search engine (See Section 3.3). With *elasticsearch-rails*[9] there is a Ruby on Rails extension available which integrates Elasticsearch into the overall MVC architecture. It is maintained by Elastic[10], the company behind Elasticsearch. Because of the good architectural fit and the strong company support, we will employ this Ruby on Rails extension to access the search engine from the web application.

[8] https://www.redmine.org/projects/redmine/wiki/Security_Advisories, last accessed at 12.01.2017
[9] https://github.com/elastic/elasticsearch-rails, last accessed at 13.01.2017
[10] https://www.elastic.co/, last accessed at 13.01.2017

The search should also feature novel visualisation techniques for its results. We decided against implementing the visualisation methods as part of the server-side stack. It would require to render the visualisations as images, allocating significant CPU resources, and delivering them to the client at high bandwidth cost. It would also make interactive visualisations, which could be directly manipulated by the user, entirely impossible. Instead the visualisations will be produced directly in browser while the server will only send the necessary data sets to the client (see Section 3.3.3).

We have implemented a prototypical integration between eScience and Elasticsearch. The final architecture and the decision on how we will integrate with the existing eScience search will be reached once we gained more experience from the usage of the prototype.

User management integration The eScience platform supports multiple forms of authentication. In addition to the in-app user management, it supports LDAP[11] and Shibboleth[12] authentication. Both are good candidates to integrate the platform into existing user data management systems. The latter one is especially popular in research institutions emerging as the leading single sign-on solution. The DFN-AAI federation, in which German research institutions using Shibboleth organise, currently lists 203 members.[13] LDAP, on the other hand, should be a good fit for the deployment in private institutions as it allows to delegate user management to Active Directory[14] or any other directory services. We expect that no further work will be necessary to fulfil this requirement.

A more fine-granular architecture will be defined later during the agile platform development process as more and more of the exact requirements emerge from WP 1, 2 and 3.

3.2.3 Responsive design

The objective of the MOVING responsive design is to make the platform accessible via a web client and functionally accessible via smartphones and tablets. So this will make sure that the two use cases which we are developing in WP1 will work smoothly on the generic MOVING platform and will be available through our target channels. For implementing the responsive design, we agreed to build the MOVING web application using the Bootstrap Framework[15]. Our decision to use this framework was based on the following rationale. The Bootstrap framework is the most popular HTML/CSS/JavaScript and free front-end framework for developing responsive, and mobile-first web applications. With Bootstrap we can easily and efficiently scale our MOVING web application with a single code base. That means we need only one framework for every device. With Bootstrap, we can use the extensive library for common HTML elements (including HTML5 elements), custom HTML and CSS components, and jQuery plugins (including visualisation of data). Given Bootstrap's immense popularity, there is a large community using the framework. It has 81,500+ stars on Github, compared to other front-end frameworks (e. g. Foundation[16] with 20,000 stars). Because of its popularity, there are more Bootstrap-based themes available. Last but not least it has good browser compatibility. Bootstrap still supports Internet Explorer 8. Having support for older browsers can make a difference, e.g. for students that use the university infrastructure for accessing the MOVING platform (which may be less likely to employ the most up-to-date browsers).

3.3 MOVING search engine

In order to sufficiently address MOVING's use cases, the platform needs to process huge amounts of text data coming from different data sources. These preexisting datasets contain different document types, e.g. bibliographic data coming from the ZBW economics dataset (see D6.2: Data management plan for an exhaustive listing and description), crawled web pages coming from Section 3.4 as well as the video transcripts. For handling these efficiently and effectively, MOVING's search engine needs to provide a scalable real-time search, support for multiple document types per index, multitenancy[17], different file formats and different programming languages. To this end, we compared several state-of-the-art search engines like Solr[18], Terrier IR[19] and Elasticsearch[20].

[11] https://tools.ietf.org/html/rfc1487, last accessed at 20.01.2017
[12] https://shibboleth.net/, last accessed at 20.01.2017
[13] https://www.aai.dfn.de/verzeichnis/idp-dfn-aai/, last accessed at 13.01.2017
[14] https://technet.microsoft.com/de-de/windowsserver/dd448614.aspx, last accessed at 13.01.2017
[15] http://getbootstrap.com/
[16] http://foundation.zurb.com/
[17] http://www.elastic.co/blog/found-multi-tenancy, last accessed at 19/01/2017
[18] http://lucene.apache.org/solr, last accessed at 20/01/2017
[19] http://terrier.org/, last accessed at 20/01/2017
[20] http://elastic.co, last accessed at 20/01/2017

Solr and Terrier IR natively support various data formats such as XML, CSV and JSON whereas Elasticsearch requires all input files in the JSON format. However, Solr and Terrier IR do not support multiple document types per index which is needed to integrate video transcripts, crawled web pages and bibliographic data into the same index. Additionally, Elasticsearch offers a more complex query syntax than Solr and Terrier IR, which allows creating structured queries beyond Lucene query syntax. Furthermore, since Elasticsearch strongly encourages adding custom functionalities to the index, extensions like custom scoring methods can be integrated more efficiently. Finally, Elasticseach provides official client libraries for Python, Ruby and Java among others which allows easy integration with all parts of our platform. Elasticsearch is known in the Big Data community to facilitate the real-time availability of data. Further high-level functionalities can be integrated using related open source products such as Logstash[21] and Kibana[22]. Logstash provides detailed analysis features of the indexed data. Kibana enables a variety of search result visualisations and is currently used for experimenting.

Although Elasticsearch does not support all required input formats, it is the only search engine that covers all other requirements. Therefore we decided to use Elasticsearch to implement MOVING search engine and provide more implementation details in the following subsection.

3.3.1 Indexing with Elasticsearch

Elasticsearch is developed in Java and is published as open source under the terms of the Apache license. The input data is required to be in the JSON format which allows handling the document meta-data differently. In Listing 1 the document has a single property named *title*. The attributes (`_index`, `_type`) are internal attributes of Elasticsearch which represents the index name and document type.

Listing 1: Simplified JSON document for indexing

```
1
2  {
3    "_index": "economics",  # Index name
4    "_type": "publication", # Doc Type
5    "_id": "10005649398",    # Doc ID
6    "_source": {
7      "title": "Expansion␣of␣the␣Money␣Supply␣with␣a␣Fixed␣Exchange␣Rate"
8    }
```

Creating an index in Elasticsearch requires creating an index configuration, which is composed of two essential parts. The first one is called *settings* and covers all aspects related to the index setup. e. g. the number of shards and replicas [23] and the data preprocessesing methods like stop word removal. The second part is called *mappings* and defines how a document and its fields are stored and indexed. Additionally, the ranking algorithms are defined in the mappings file. Below, we describe how we configured the initial index in the MOVING platform.

Settings The index configuration is a YAML file. An example comprising the settings and the mappings is presented in Listing 2.

Listing 2: Index settings structure

```
1  settings:
2    index:
3      # basic index settings
4    analysis:
5      filter:
6        # token filter settings
7      analyzer:
8        # analyser settings
9  mappings:
10   # mapping definition
```

[21] http://www.elastic.co/guide/en/logstash/current/getting-started-with-logstash.html, last accessed at 19/01/2017
[22] http://www.elastic.co/products/kibana, last accessed at 19/01/2017
[23] http://www.elastic.co/guide/en/elasticsearch/guide/current/_index_settings.html, last accessed at 19/01/2017

Mappings The mappings define which information of the data is indexed and how it is processed. Our first running prototype is restricted to the title property. The title is split into multiple fields which allows multiple analyses on the same data and comparison of different retrieval models like TF-IDF (Salton & Yang, 1973) and CF-IDF (Goossen, IJntema, Frasincar, Hogenboom, & Kaymak, 2011). A detailed survey of the possible retrieval models will be part of the upcoming deliverable (D3.1: Technologies for MOVING data processing and visualisation). In Listing 3 we present a simplified version of the mappings used in our prototype.

Listing 3: simplified version of the mappings

```
mappings:
  publication:
    properties:
      title: title
      type: string
      analyzer: default
    fields:
      TFIDF:
        type: string
        analyzer: TermAnalyzer
        similarity: default
      CFIDF:
        type: string
        analyzer: ConceptAnalyzer
        similarity: default # TFIDF
```

Analysers Analysers are required before the actual mapping can be done. Elasticsearch provides a large variety of analysers which could be applied to documents before indexing. For instance the *Stop Analyser* in Elasticsearch supports removal of stop words like *the* and *an* from the documents. Stop word removal together with stemming are fundamental preprocessing tasks well known in the text mining community. However, it is possible to define custom analysers using the YAML syntax. In Listing 4 we present the configuration for our concept analyser. With a combination of provided and custom analysers as well as custom token filters we implemented our novel approach HCF-IDF (Nishioka & Scherp, 2016).

Listing 4: Concept analyser

```
analyzer:
  TermAnalyzer:
    type: custom
    tokenizer: standard
        filter:
                [english_possessive_stemmer, lowercase,
                 english_stop, # english_keywords
                ,english_kstem]
  ConceptAnalyzer:
    type: custom
    tokenizer: standard
        filter:
                [english_possessive_stemmer, lowercase,
                        alt_labels, # resolve altlabel synsets
                        pref_labels] # keep only preflabels
```

Filters Filters are usually implemented as token filter[24]. Token filters differ from analysers in terms that they use non-default values. One may specify token filters with customised options for later use. As an example we take a look at the *alt_labels* and *pref_labels* token filters, which provide synonym and keep words [25] (shown in Listing 5), respectively.

[24] http://www.elastic.co/guide/en/elasticsearch/reference/2.3/analysis-tokenfilters.html, last accessed at 19/01/2017

[25] http://www.elastic.co/guide/en/elasticsearch/reference/current/analysis-keep-words-tokenfilter.html, last accessed at 19/01/2017

Listing 5: Configurations for replacing some labels with others

```
token filters:
  alt_labels:
    type: synonym
    expand: false
    synonyms_path: analysis/altLabels.txt # relative to config dir
  pref_labels:
    type: keep
    keep_words_path: analysis/prefLabels.txt # relative to config dir
```

These two token filters make use of external text files which hold the synonyms or the keep words respectively.

3.3.2 Adding data into Elasticsearch

The Elasticsearch index provides certain incremental index computation features for updating the index. The examples below reference a JSON file called `document.json`, which consists of one document (Listing 1).

REST API The REST API allows performing different operations. For instance, integrating new data using the standardised HTTP protocol (Listing 6).

Listing 6: Simple HTTP call for adding data

```
curl -XPUT localhost:9200/_bulk --data-binary @document.json
```

Python API The Python API is a wrapper for the REST API, callable within Python programs (Listing 7).

Listing 7: Simple Python call for adding data (The variable `JSON` contains the example JSON file presented above)

```
es.index(index='economics', doc_type='publication', id=i, body=JSON)
```

3.3.3 Search result visualisation

To visualise search results, potentially existing further entities, and links between them, we currently develop the *Graph Visualization Framework* (GVF). It will provide interactive possibilities of deep visual analytics. It is based on the *ANGULAR 2*[26] framework, written in *TypeScript*[27] (ES5), compiled to plain *JavaScript* code. Since its basic concept is to support multiple graphs, which are interactively connected in one framework, it holds a dynamic number of HTML canvas instances, each of which is used to render a *THREE.js*[28] scene. This library is a framework for the *WebGL*[29] API, used to visualise complex scenes, supported by the user's graphics processing unit acceleration.

The high-level language character of TypeScript allows to usage of well designed object-oriented programming. Thus, it the framework will be extended with custom visualisations by deriving existing ones and adding them in a plugin environment we currently plan.

Evaluating possibilities to store, load and retrieve visualisation data is currently in progress. A local storage of data in the browser's *DOM Storage*[30] should be possible, as well as a dynamic XHR-based[31] communication with a data providing server, where the results and other entities are stored in a database. Setting the data via event calls should also be possible after loading them.

Integrating our framework into the MOVING web application is also currently under discussion. We consider whether it is possible to load it as a JavaScript library or to use it inside a HTML inline frame[32]. However, communication between the MOVING web application and GVF will happen via different well defined events.

[26] https://angular.io/, last accessed at 17.01.2017
[27] https://www.typescriptlang.org/, last accessed at 17.01.2017
[28] https://threejs.org/, last accessed at 17.01.2017
[29] https://www.khronos.org/registry/webgl/specs/latest/2.0/, last accessed at 17.01.2017
[30] https://www.w3.org/TR/webstorage/, last accessed at 17.01.2017
[31] https://www.w3.org/TR/XMLHttpRequest/, last accessed at 17.01.2017
[32] https://www.w3.org/TR/html5/embedded-content-0.html#the-iframe-element, last accessed at 17.01.2017

3.4 MOVING crawler

The MOVING crawler is responsible for the collection of data from the internet and its delivery to Elasticsearch for indexing. There are three main sources we want to crawl data from:

- social media,
- specific websites
- and the web in general.

For each of the above sources, we have one dedicated sub-crawler: the stream manager, the focused crawler and the web search.

3.4.1 Focused crawler

The focused crawler's task is to collect data from specific websites. Although the task is simple, tools like the build-in GNU `wget`[33] lack the functionality to selectively download specific types of data (declared on the header's `content-type` attribute) or respect the website's `robots.txt`, that defines their crawling policy. So, unavoidably, we had to choose a more refined tool for the task. The focused crawler is based upon the Scrapy Python web crawling/scraping framework[34]. The reason for choosing Scrapy was that it is a fast and powerful framework that also allows the programmer full freedom to customise the crawling process. With Scrapy we can choose which links to follow or not to follow, based on their content type and file extension.

The Scrapy framework uses crawling units called spiders. The spider starts from a given set of URLs, and follows their links to other URLs recursively, until it has crawled the whole domain. Since the purpose of the focused crawler is to crawl many different domains we have one general purpose spider for all the domains. The spider extracts the data from the HTML pages, using built-in Scrapy XPath extractors, and sends them to a pipeline. The pipeline is the Scrapy component responsible for the processing of the data extracted. Our pipeline sends the data to Elasticsearch for indexing using its REST API. The data model is the same for all the crawlers. The focused crawler is set to obey the `robots.txt` of the websites (if it exists), and has an average default download delay of 1 second for politeness. Many spiders (each for a different domain) may run simultaneously in different threads. The websites to be crawled are inserted using a Python script (input/domains.py) to the MongoDB collection `Domains`.

The focused crawler uses a scheduler to check at predefined intervals for new domains inserted by the users of the platform. If there is a new domain, it launches a spider to crawl it. The current version has a standard revisit policy of 30 days.

3.4.2 Web search

The web search periodically searches the web for specific topics. Since the development of a decent web search engine for the whole web from scratch requires huge distributed computation power and storage capacity, we decided to use the APIs of existing search engines. One API with free unlimited access is the FAROO REST API, but since it infrequently re-indexes the web, the results are usually outdated. Also, Yahoo's web search API has been deprecated and Bing web search API offers very limited free access. That leaves us with the Google Custom Search API provided freely by Google for up to 100 requests per day, which translates to 1000 URLs per day. It is important to note that web search is a component of the MOVING crawler and should not be confused with the MOVING search engine (Section 3.3). The data retrieved through web search is then ingested to the MOVING search engine and harvested through that.

To use the Google Custom Search API we have created a Google account for the MOVING project and created an API key. Our search engine configuration defines the domains to be searched. which is the entire web. The web search uses a scheduler that runs once a day until the maximum daily requests are reached or all the topics have been fully searched. For each topic the Google Custom Search API can retrieve up to 100 URLs. As with the focused crawler, the current web search version supports only HTML results. The topics to be searched are inserted to the MongoDB collection `Collections` using a simple Python script (`input/topics.py`). Data is extracted from the HTML pages and then sent to Elasticsearch for indexing using its REST API. The data model is the same as with the focused crawler.

[33] https://www.gnu.org/software/wget/
[34] https://scrapy.org/, last accessed at 23.01.2017

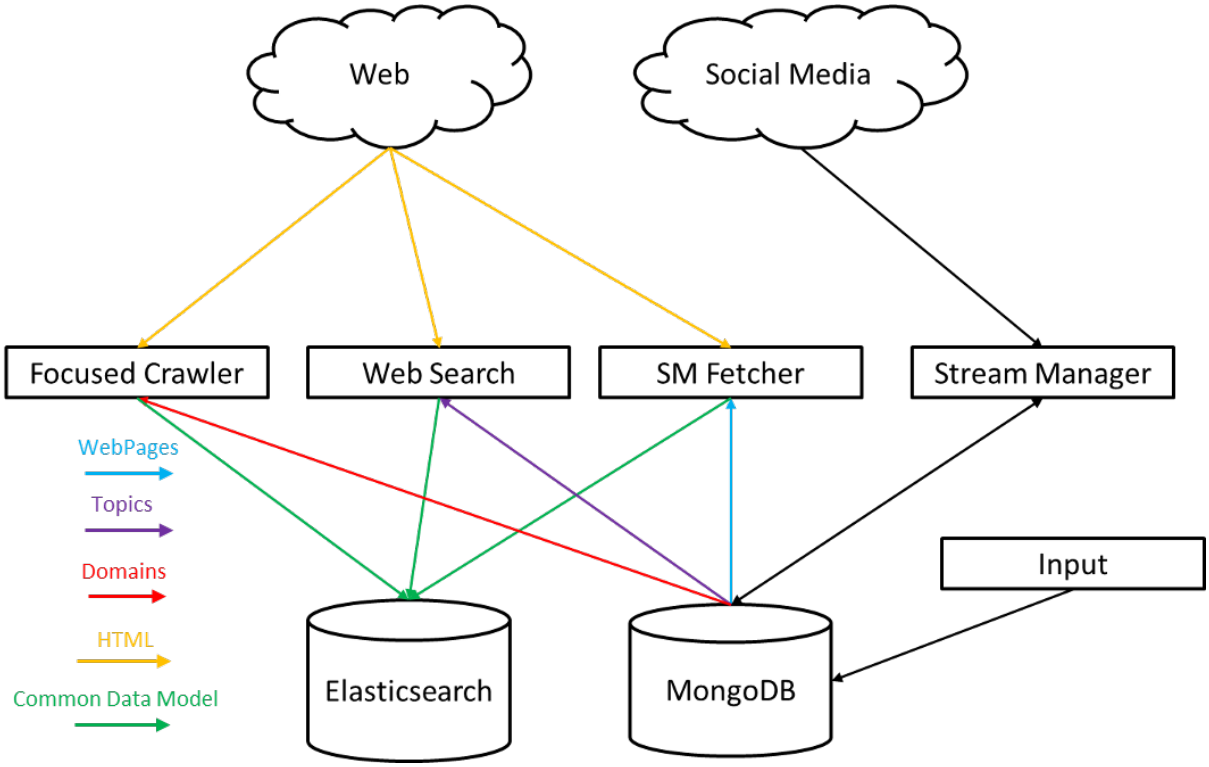

Figure 2: The architecture of the MOVING crawler

3.4.3 Stream manager

For the task of collecting data from social media we use the stream manager, a software that meets the requirements perfectly. It is a Java open source project that was developed for the FP7 project SocialSensor[35]. For MOVING, we use its functionality for three social platforms: Twitter, YouTube and Google+. It is hosted on GitHub[36] and uses Maven for the build process. In order to get the feeds from the social platforms, we have created accounts with API keys for MOVING for all the platforms. The stream manager can collect feeds based on user accounts, keywords and location. In the current version we only use keywords, but it can be easily extended. The keywords are equivalent to the topics used by the web search and are inserted the same way. The stream manager stores the collected data in MongoDB in several collections:

`Item` posts, tweets

`MediaItem` `Items` which have multimedia content

`WebPage` `Items` with webpage links

`StreamUser` Social platform users who have created or are mentioned in `Items`

The stream manager may store the `Items` with web page URLs, but it does not store the actual web page. This task is taken over by the stream manager fetcher (SM fetcher). The stream manager fetcher is a simple Python script that uses a scheduler to periodically check for URLs retrieved by the stream manager, it downloads the HTML, extracts the data and sends them to Elasticsearch for indexing using its REST API.

3.4.4 Input

The topics and domains to be crawled by the three crawlers are inserted using simple command line Python scripts (`input/topics.py` and `input/domains.py`) and passing the topics (or domains respectively) as arguments. For the detailed MongoDB schema see Appendix A.

[35] http://socialsensor.eu/
[36] https://github.com/MKLab-ITI/mklab-stream-manager

Topics This collection contains the topics (or keywords) that will be used by the stream manager and the web search as inputs. It can also contain social media accounts and locations for the stream manager to monitor.

Domains This collection contains the domains that will be crawled by the focused crawler e.g. `bayer.com`.

3.4.5 Data model for HTML pages

The following list contains the attributes that constitute the data model used by the crawlers. The data is sent in JSON format to Elasticsearch for indexing.

URL The URL of the webpage

Title The title of the webpage `<title>`

Abstract An abstract description of the webpage `<meta name="description" content />`

Fulltext The raw contents of the HTML file

Authors The author of the webpage. `<meta name="author" content />`

Date start The creation time or the last time the webpage has been modified. The value of `last-modified` header attribute. If not present, current date and time is inserted.

Date end Same as `Date start`

Venue The domain name of the website

License The license of the web page's content (if present) `<a rel="license" />`

Language The language of the webpage <html lang>

Keywords A list of keywords (tags) for the webpage `<meta name="keywords" content />`

3.5 User interaction tracking and dashboard

Users will consent to have their interactions tracked. The first time users access the platform, they will be shown information detailing the purpose of the tracking. A link with further details about the nature of the captured data will also be provided. If the user agrees, a JavaScript code will be loaded every time the user accesses any page on the platform. An option to revoke the permission to capture the interaction will be available in the profile section of the web application options.

3.5.1 User interaction tracking

UCIVIT[37], a tool developed at the University of Manchester, will be employed to track user interaction. It requires to add a piece of JavaScript code to the web pages to track, and it automatically captures users' interaction data, storing it in a MongoDB database. The tool is well suited for the MOVING platform, as the necessary code can be dynamically loaded into the page, and it does not cause noticeable overheads to the interaction. The tool will be modified to tailor the capturing of specific interaction data to further support the interaction dashboard.

Specific users are tracked via a unique, anonymised code stored in a cookie. Captured events include all mouse and keyboard interactions, as well as browser window events, changes to the state of the elements on the page, and other system information. The state of the Document Object Model (DOM) and its changes during interaction are also stored, allowing the context of interaction to be potentially recreated. All the data is stored in JavaScript Object Notation (JSON) format in a Non Structured Query Language (NoSQL) database, which allows for extensibility and query scalability, and means that further events can easily be included as required. The design also supports the modification or removal of existing events should they become deprecated.

Window events, indicating when the page has finished loading or when the page has lost focus, combined with user specific information, provide context for lower-level events. This information allows the grouping of events according to the specific environment in which they took place, such as the browser tab – identifiable

[37] `https://github.com/aapaolaza/UCIVIT-WebIntCap`

via the page load timestamp – and URL. Logging the platform and browser version supports the control of event compatibility according to a system. The ever-changing nature of the web and the possibility of enabling partially supported events makes this information useful for discerning problems with the way a given system handles interaction events.

- **Web site ID**: identifier of the web domain generating the data (e.g. *10002*, *10006*).
- **User ID**: identifier for each user, stored in a cookie (e.g. *yAubjmZKSRd9*, *ITPRPmVwHJLo*).
- **IP**: user's Internet Protocol address (e.g. *130.88.99.220*).
- **URL**: address of the web page (e.g. *moving-project.eu*).
- **Timestamp**: time stored in ms based on the capture server (e.g. *1448466686*).
- **Load timestamp**: time when the page finished loading, same for all events occurring within that page (e.g. *1537250757*).
- **Time offset**: difference in minutes of the user's local time with respect to the Coordinated Universal Time (UTC) (e.g. *0* for UK, *+60* for Spain).
- **Platform**: details about the operating system (e.g. *WinNT, win32, linux_x86*).
- **Browser**: details about the browser (e.g. *Chrome46.0.2490.86., Firefox 42.0*).

Certain low-level events (see Table 1) are a combination of events automatically triggered by the browser during user interaction, such as *Mouse down* and *Mouse up*, and manually triggered ones. In the case of automatically triggered events, the extent to which they function correctly depends entirely on the browser. Some of the events – marked with an asterisk (*) in Table 1 – are only partially supported by a few browsers and are therefore captured for possible use, but usually discarded in subsequent analyses to avoid an inaccurate interpretation of an unsuccessful interaction event.

Manually triggered events rely on periodic queries at predefined intervals or queries triggered under certain conditions. To obtain the value of *Window scroll*, the state of the browser's window scroll is queried every 200 ms. In the case of *Mouse move*, the position of the mouse is compared every 150 ms. A *Mouse select* event is triggered at every *Mouse up* event, on checking if the result of that action is the selection of any content.

For some events, additional information is captured. *Mouse coordinates* are stored for events concerning mouse interaction. The reported *offset* coordinates are relative to the web page's node element where the interaction is taking place. The stored information is the following (the example has coordinates '243x204' and offset coordinates '31x8'):

- **X coordinates** X coordinates relative to window (e.g. *243*).
- **Y coordinates** Y coordinates relative to window (e.g. *204*).
- **OffsetX coordinates** X coordinates relative to node element (e.g. *31*).
- **OffsetY coordinates** Y coordinates relative to node element (e.g. *8*).

Additional events that store other event-specific information include: the pressed key for *Keyboard* events; content of the selected text for text selection events; and a measurement of the scrolled distance for scroll events. If any event imply interaction with a node, information about that node is also stored. The following node information is captured to allow identification of the element the user is interacting with. An asterisk (*) indicates that this value is only captured if the DOM element contains this specific attribute. In the case of text value, it is only captured if the node is of type text.

- **DOM path** path of the node in the DOM (e.g. *BODY/DIV[5]/TABLE[1]/DIV*).
- **Id** id attribute of the node* (e.g. *searchButton* from node *<button id="searchButton">Search</button>*).
- **Name** name attribute of the node* (e.g. *div* from node * text *).
- **Type** same behaviour as name, added for increased compatibility.

Type	Events	Additional information	Node information
Mouse	Mouse down	Mouse coordinates	✓
Mouse up	Mouse coordinates	✓	
Mouse move	Mouse coordinates	✓	
Mouse over	Mouse coordinates	✓	
Mouse out	Mouse coordinates	✓	
Double click	Mouse coordinates	✓	
Mouse wheel	Amount of scroll	✓	
Selection	Select event*	Selected text content	✓
Mouse select	Selected text content	✓	
Cut*	Selected text content	✓	
Copy*	Selected text content	✓	
Paste*	Selected text content	✓	
Keyboard	Key down	Pressed key	✓
Key up	Pressed key	✓	
Key press	Pressed key	✓	
Window	Load	Window size	
Resize	Window size	✓	
Unload			
Window focus			
Window blur			
Scroll	Amount of vertical or horizontal scroll		
Focus change	Focus		✓
Blur		✓	
Miscellaneous	Change	Element type and new value	✓
Context menu	Mouse coordinates	✓	

Table 1: Events captured in desktop web browsers

- **Href** destination of the link contained in the node* (e.g. *url.com* from node *link text*).
- **Text content** first 100 characters of the text content (e.g. *text sample* from node * text sample *).
- **Text value** text value of the node* (e.g. *textInput* from *<input value="textInput">*).

Mobile events are also captured. Any processing of the captured events on the client side has been avoided. Therefore, composite events common to touch interfaces as pinch, swipe, and drag events have not been considered. Instead, all events regarding touch input have been captured, including context information such as the number of fingers in contact with the screen. This way extraction of complex touch gesture interactions is possible. All the mobile events contain the basic information previously mentioned, as well as additional information, such as the coordinates of the touch event and the number of concurrent touch inputs.

3.5.2 Interaction dashboard

As a way to ease the access to the captured data, an interaction dashboard will be implemented. The *MOVING recommender* systems will employ this dashboard to retrieve interaction information about the users. This

way recommendations can be tailored, and individual user's progress throughout their knowledge acquisition can be tracked.

The dashboard will provide a tool to design and extract sequences of the captured events. Analysis of these sequences allows designers to test a hypothesis regarding users' interaction. For example, the designer might believe that users scroll less over time. If so, information outside the initial view of the page would remain obscured. Sequences of scroll interaction can be extracted for analysis of their occurrences over time. The frequency of these sequences, as well as various attributes (such as distance and speed, in the case of the mentioned example) can be analysed. An interactive web application to construct these queries has been built. A graphical notation can be used to define the interaction event sequences and patterns designers are seeking including temporal relations between events, supporting the creation of complex queries.

A series of use cases will guide the design of the MOVING platform. However, the real use of the platform might differ from the designers' expectations. Users' workflows will be extracted from the captured interaction data, to be compared to the initial designs. A combination of visual exploration and sequence mining will be combined to support the exploration of the workflows.

The designer will be able to design the sequences that conform to the workflow. These sequences can be composed of both high-level events, identified from the MOVING platform (such as relevant interactions with buttons), and event sequences extracted following the methodology described in the previous section.

Extracted sequences from the users interaction can then be compared to the projected workflow in a visual manner. Outliers deviating from the projected interaction can be easily identified.

Additionally, sequence mining techniques will be employed, to identify outliers from completely different workflows. Once these workflows are defined, they can be detected in real time during users' interaction. This contributes to validation of the use cases, providing the means to obtain remote feedback. Validating that the platform complies with the specified requirements will be possible using the provided dashboard.

3.5.3 Implementation of the adaptive training support

The adaptive training support will be implemented as a widget, that is a light-weight and flexible representation way that can easily be integrated into a web user-interface. The architecture of the adaptive training support will consist of a client-server architecture.
For the client, we will use HTML5, JavaScript and CSS to represent the ATS visualisations and prompts. The client has three main tasks: i) it will receive the data sent from the adaptive training support engine, ii) it will determine how to visualise the data, and iii) it will determine when to visualise the data. The decision for a feasible HTML5 library has not been taken yet.
On the server-side, we will implement an adaptive training support engine that will consist of the following three building blocks: i) a data gathering tool, ii) an analysis tool and iii) a database.

- Data gathering tool: the engine will gather data captured and provided by the MOVING platform. specifically, it will use data stored in the user models, data captured and stored in the MongoDB of the user interaction tracking tool and data provided by the MOVING recommender.
- Analysis tool: the tool will analyse the collected data to detect trigger interaction patterns and information worth being reflected on. Furthermore, it will convert the data into a JSON-encoded data string for the communication with the client.
- Database: The database will store all analysis done by the analysis tool. In addition it will store the information created by the user e.g. a reply to a prompt. The decision, which database will be used, has not been taken yet.

The communication between the client and the server will be established with REST API calls. The data will be sent in a JSON-encoded format.

4 Feasibility studies for use cases

In this section present how the use cases can be implemented with the chosen MOVING architecture.

4.1 Use Case 1: Research on business information by public administrators

The first use case has been coordinated by Ernst & Young and describes the usage of the MOVING platform for the research on business information by public administrators. It presents two scenarios to support the use case:

Research on compliance to European laws and regulations Any organisation, research institutes, universities and companies likewise need to be compliant with the increasing number of current laws and regulations. In this scenario, the compliance officer Mr. Clark has been assigned the task to identify potential risks how the economic and financial changes in law and regulations in the European market may affect the organisation's compliance. As part of his research, Mr. Clark is using the MOVING platform to conduct an analysis of the political, economic, social and technological factors that has to be performed. This "PEST-Analysis" is usually performed as part of a "SWOT-Analysis", identifying strengths, weaknesses, opportunities and threats.

Innovation in advisory services The success of professional advisory services is highly dependent on identifying trends and innovations and being trained in transforming those into solutions for different market sectors, industries and clients. With the MOVING platform users will be able to perform broad and flexible searches for those trends and think "out of the box".

The use case identifies a set of key features helping compliance officers and professional advisors to achieve their goals:

Topic-based filtering The user selects from predefined filters in which areas the search has to be performed. These are political, economic, social and technological. Multiple selections are possible as well as subtopics, e. g. economic might be split into macro-economical, tax-related, business administrative, managerial accounting and compliance topics. Let us say that the user decides to concentrate on the macroeconomic category.

Faceted search The user may determine to limit the search to certain countries (where the publication has been made) or the language of those publications. Further filtering in regard to a date range, length of publications, type of publication, industry and ratings of other users may apply as well. The user might decide to concentrate on European countries, publications within the last 5 years, professional journals with a minimum length of 3 pages and published in English, French and German.

Tag cloud As a first view the user will see a tag cloud presentation of all topics identified by the MOVING platform. The size of the tag is determined by the importance of the topic. This importance may be based on the frequency of words, i. e. number of publications containing these words. At any stage, the user will be able to drill down into the individual publications that are available as full text. We expect that topics like credit crisis, tax evasion, nonperforming loans, market barriers, sourcing and emerging markets will appear.

Text network As an alternative analysis instrument the MOVING platform will provide a text network. The user can also enable semantic analysis of a search result and view it as a topic network highlighting the specific economics subjects and their relations. This network view will not only show the importance of individual topics but also the strength of the connection between different topics. The user may drill down into a topic which will then show a sub-network on keywords inside a topic.

Lifecycle view To be able to perform a risk-based prognosis, the compliance officer will need to see the evolution of topics over time. To enable this, the MOVING platform will contain a lifecycle view for a selected topic. With an adjustable time frame, the measures above will be used to see when a topic appears, how it develops/evolves and maybe degenerates over time. The user in the compliance scenario may determine that sourcing was an issue in certain areas and industries in the past but is no longer critical to European business locations. For other topics, e.g. tax planning this might be different which is visible as the topic is still evolving.

The architecture presented in this document is not yet detailed enough to show how these features can be implemented in the MOVING platform. At this stage we can only map the features to the relevant components in the architecture. The concrete specification of these components will be designed in the individual work packages:

- The topic-based filtering and the faceted search will be defined and implemented in the MOVING search engine (see Section 3.3). For the topic search we will maintain a list of topics for each indexed document. The extraction process of the topics from the documents will be developed in WP3. The faceted search will rely on a set of indexed metadata attributes which the MOVING platform can query for. These attributes will be defined in the common data model (see Section 3.3). The user interface for the search masks will be implemented in the MOVING web application (see Section 3.2.1) after their appearance has been decided in WP1.

– The tag cloud, text network and the lifecycle view are instances of search result visualisations (see Section 3.3.3) we intend to develop for the MOVING platform in WP3.

4.2 Use case 2: Managing and mining research information

The second use case provided by TU Dresden specifically addresses the information and training needs of young researchers like PhD students and Master's theses writers. It presents four scenarios highlighting different stages of a research project in which the MOVING platform can be utilised.

Scenario 1: State-of-the-art on a research topic

This scenario describes how a user ("Ms. Brown") interacts with the search feature of the platform to get an overview about a research topic she has little or no previous knowledge about. She starts with very broad and unspecific search terms ("Internet of Things"). She is then dissatisfied with the search results, because there are more than she can grasp. Because of her little knowledge about the topic she has trouble narrowing down the search on their own. Then she discovers the adaptive training support (ATS) widget (see Section 3.5.3). It helps her to successfully use the topic-based filtering and the faceted search (see previous use case in Section 4.1) to limit the search to a manageable result list. The same widget also introduces her to the search visualisation feature (see Section 3.3.3). Before she enables the search result visualisation, she familiarizes her self with the feature by watching the linked tutorial video about the feature. This video can be a part of the curricula (see Section 3.2) of the MOVING platform. She learns that she can use the search result visualisation to get the connections between the search results. This way she can discover the structure of her research topic and limit the search to the parts she is interested in. Then she activates the lifecycle view (see previous use case in Section 4.1). This time she got the hint from a wiki page (see Section 3.2.1) created by other users of the platform after the ATS widget has suggested this wiki page as relevant to her search.

Scenario 2: Finding suitable partners for research projects that are active in the respective research fields

In this scenario the user (still "Ms. Brown") visits the MOVING platform to find prospective research partners. She starts from the original search she has performed to get an overview about her research topic. From there, she switches the search mode from documents to persons instead. In the text network view (see Section 4.1) she can see which persons are linked to which topic. She then selects the desired topic and gets a list of relevant researchers. At this point the adaptive training support widget shows her the recommendation to use MOVING platform to contact these researchers. This can be achieved by sending a message to the researches who are registered on the platform. This is one of the features the eScience platform will bring to MOVING (see Section 3.2.1).

Scenario 3: Strategic decision for research funding

Scenario 3 describes another interaction of the user with the search function and search result visualisation techniques of the MOVING platform. After Ms. Brown now has a project consortium, she and the other researchers need an additional financial budget to work on the research question. She again uses the MOVING platform to gain information about an unfamiliar topic. This time she searches for funding strategies in the context of her research topic. Eventually she discovers that crowd funding would be good fit for her. In this process she interacts with the same features as previously mentioned in scenario 1.

Scenario 4: Accompanying training materials, courses and tutorials

In the last scenario the user employs the MOVING platform to learn more about the funding method she is interested in ("crowd funding"). She starts again with her last search for funding methods but chooses to search for training material instead. She finds a cMOOC related to the topic hosted on the platform. cMOOCs support will be implemented in the MOVING web application according to the requirements from WP1 (see Section 3.2.1). She also recommends the platform to her students. When her students visit MOVING platform, they find tutorials explaining the purpose and usage of the platform. These tutorials from WP2 can be deployed on the platform as wiki pages (see Section 3.2.1) and suggested by the adaptive training support widget (see Section 3.5.3).

5 Supporting software

This section lists all software and tools used for the development process and for the quality assurance of the MOVING project.

5.1 Source code management

Source code management is achieved in the MOVING project with the popular source code management (SCM) software *Git*[38]. There is a main repository for the MOVING web application and a few other supplementary repositories for other software artefacts and resources. Table 2 lists all repositories in use with their purpose. More repositories can be created in the future if the need arises.

Name	Description
`moving`	The main repository hosting the MOVING web application.
`movingbox`	This repository contains the configuration files for the shared development environment.
`moving-crawler`	The application used to gather information from various sources and feed it into the Elasticsearch index is developed in this repository.
`moving-documents`	Documents created during the MOVING project for which version control makes sense (e. g. Latex documents) are stored here.

Table 2: MOVING repositories

All repositories are hosted at the FusionForge service[39] kindly provided by the TU Dresden. FusionForge is an integrated development platform for software projects. The MOVING project uses the service to enable secure access to its Git repositories for the consortium members.

5.2 Development environment

In order to provide a common development environment for all technical partners, preconfigured virtual machines (VM) are utilised. The configuration for the VMs is performed by the open-source software Vagrant[40] by HashiCorp. VMs managed by Vagrant are so called *boxes*. Vagrant uses a single text file written in Ruby (named "Vagrantfile" by convention) to describe fundamental properties (e.g. disk and RAM size), the operating system being used and additionally how the system is provisioned. This *provisioning process* (installing software and configuring it) is performed by the open-source software Ansible[41]. it is described declaratively with text files written in the language YAML and in Jinja templates, whose aggregation is called Playbook.

By using these two core technologies the MOVING development gains the following benefits:

- setup and maintenance of development machines require just minimal effort (only few commands needed; for details see `README.md` from the movingbox repository),
- all configuration artefacts are presented as text files and thus are fitting for any SCM software (allowing better sharing and tracking of incremental changes),
- improved interopability between invidivual partners' contributions by having a common base and
- configuration artefacts can be re-used in later deployment (see Section Deployment).

Vagrant interacts with so called *providers* that are responsible for performing the required changes for the VM issued by Vagrant. Out of the box supported providers are VirtualBox, Hyper-V and Docker. Possibly more providers can be supported[42]. For now, VirtualBox[43] is used for the development of the MOVING platform.

[38] https://git-scm.com/, last accessed at 23.01.2017
[39] https://fusionforge.zih.tu-dresden.de, last accessed at 23.01.2017
[40] https://www.vagrantup.com/, last accessed at 05.01.2017
[41] https://www.ansible.com, last accessed at 05.01.2017
[42] https://www.vagrantup.com/docs/providers/, last accessed at 05.01.2017
[43] https://www.virtualbox.org/, last accessed at 05.01.2017

5.3 Deployment

The deployment and maintenance of the MOVING platform is intended to be as fail-safe and as easy as possible.

Typical tasks in that domain involve performing system updates, installing new software, creating new and changing existing configuration files and controlling various services on the machine.

Compared to manual administration of servers, the chance of running into errors can be significantly lowered by using IT automation such as Ansible.[44] It supports bundling repetitive steps into reusable, parametrisable tasks (thus eliminating the likelihood of typos and forgetting important steps), and the instancing/updating of concrete configuration files via Jinja templates and a few variation points expressed with variable files written in YAML.

Thus, we use Ansible both for provisioning the productive machines and Vagrant boxes. The requirements for Ansible on the server-side are very light, as it only requires to be able to build a connection via SSH (hence only OpenSSH needs to be installed on the to-be-managed systems). This is very light-weight compared to other solutions (e.g. Puppet, Chef, etc.) that require some sort of service running on the managed system (see page 5 in (Red Hat, Inc., 2016)).

Ansible has also a concise documentation[45], that can be read in a short amount of time making it possible for newcomers in the project to quickly understand and extend existing reusable tasks.

5.4 Continuous integration

We use a Jenkins server to perform continuous integration for the MOVING platform. We implemented the following tasks:

Test Suite Execution The test suite of the MOVING web application is executed if a developer pushes changes to the repository. If one or more tests fail, an email with the error log will be sent to the developers. This gives the developers early feedback if they introduce a regression.

Deployment to test server Each night the current development state is deployed to the test server. The task uses Ansible (see Section 5.3) to achieve this. This way the test server always reflects the latest development results.

6 QA measures

We implement the following technical and organisational quality assurance methods:

Automatic regression testing The MOVING web application has an extensive suite of unit, functional and integration tests. It covers the complete application. We use this test suite for automatic regression testing. This way we can detect the introduction of bugs early on and we can act quickly to fix them.

Continuous integration We use a Jenkins server to perform continuous integration. We run the test suite of the MOVING web application and run automatic deployment to the test server. This allows the consortium members to always have access to the latest development state of the MOVING platform.

Issue tracker Part of the TU Dresden's FusionForge service is an issue tracker. We use it to efficiently report, assign and fix bugs in the MOVING platform.

Integration camps In conjunction to the regular project meetings we host so called *Integration camps*. They are one day events the project developers attend in order to find technical decisions and perform architecture reviews. This makes it more probable to find weaknesses in the design and helps to create an architecture meeting the needs of all project partners.

User studies We will conduct user studies to validate whether the provided functionality conforms to the expectations of our users. We will analyse the collected log data of the user interaction tracking (see Section 3.5) in a hypothesis-driven fashion in order to identify problematic situations or mismatches between the interface and the mental model of the users. These studies will be conducted in parallel to the development of the platform to aid the agile development process.

[44] https://www.ansible.com/, last accessed at 23.01.2017
[45] https://docs.ansible.com/, last accessed at 12.01.2017

7 Conclusion

This report dissects the architecture of the upcoming MOVING platform into single components explaining in detail how the given requirements and partner's experience lead to already concrete design decisions and pending choices of only few considered candidates. An in-depth analysis is given of how the MOVING web platform will emerge from the existing eScience platform utilising its capabilities given by the extension points from the frameworks Ruby on Rails and Redmine (compare Section 3.2.1). One of the findings in this analysis was to refrain from the usage of plugins, but rather extend the core of eScience directly. This report also covers the already agreed on technologies and establishes the initial data structures that serve both for technical and user interfaces; e.g. Elasticsearch as common search solution, see Section 3.3, or Bootstrap for the user interface, see Section 3.2.3. Vital information of the inner workings of the planned components were also presented, concerning the search (Section 3.3), the MOVING crawler (Section 3.4), the user interaction tracking and dashboard (Section 3.5).

These gathered refinements give an important push towards the upcoming implementation for the technical partners. As surely more requirements will follow, such as from deliverable D1.1 (User requirements and specification of the use cases, due on March 31 2017), this report will evolve over time. But for now, this report provides a good basis each partner can build upon.

References

Goossen, F., IJntema, W., Frasincar, F., Hogenboom, F., & Kaymak, U. (2011). News personalization using the cf-idf semantic recommender. In *Proceedings of the international conference on web intelligence, mining and semantics* (p. 10).

Krasner, G. E., & Pope, S. T. (1988, August). A cookbook for using the model-view controller user interface paradigm in smalltalk-80. *J. Object Oriented Program.*, *1*(3), 26–49. Retrieved from `http://dl.acm.org/citation.cfm?id=50757.50759`

Nishioka, C., & Scherp, A. (2016). Profiling vs. time vs. content: What does matter for top-k publication recommendation based on twitter profiles? *arXiv preprint arXiv:1603.07016*.

Red Hat, Inc. (2016). The benefits of agentless architecture. Retrieved from `http://cdn2.hubspot.net/hub/330046/file-479013288-pdf/pdf_content/The_Benefits_of_Agentless_Architecture.pdf?t=1390852839000` (Whitepaper)

Salton, G., & Yang, C.-S. (1973). On the specification of term values in automatic indexing. *Journal of documentation*, *29*(4), 351–372.

A Stream manager: MongoDB schema

Item

`id` The ID of the `Item`

`source` The name of the stream that an `Item` comes from e.g.g Twitter

`title` Title of the post or the tweet itself (in case of twitter)

`tags` List of hashtags in the `Item`

`uid` The ID of the `StreamUser`

`mentions` List of IDs of mentioned `StreamUsers`

`links` List of URLs contained in the `Item`

`publicationTime` The publication time of an `Item` (Unix timestamp)

`language` The language of an `Item`

MediaItem

`id` The ID of the `MediaItem`

`url` The url of the `MediaItem`

`thumbnail` The thumbnail of the `MediaItem`

`source` The name of the stream that an `MediaItem` comes from e.ġ. Twitter

`uid` The ID of the `StreamUser`

`title` Textual information of the `MediaItem`

`tags` List of hashtags in the `MediaItem`

`type` The type of a media item. Can only be image/video

`publicationTime` The publication time of a `MediaItem` (Unix timestamp)

`views` The number of times this media has been viewed (for videos)

`width` The width of the `MediaItem`

`height` The height of the `MediaItem`

WebPage

`id` The ID of the `WebPage`

`url` The URL of a `WebPage`. This is usually a short URL

`date` The date that a web page shared for the first time (unix timestamp)

`source` The name of the stream that an `WebPage` comes from e. g. Twitter

StreamUser

`id` The ID of the `StreamUser`

`userid` The internal ID of a user in a specific social media stream

`username` The username of the user in a specific social media stream

`source` The name of the stream that a user comes from e. g. Twitter

`items` The number of `Items` posted by the user

`profileImage` The profile image of a user

`url` A URL of the personal web page of the user

`description` A short description (bio) of the user

`createdAt` The date that this account has been created (unix timestamp)

`location` The location associated with a user

`verified` An indicator whether the user is verified by the social media service

`mentions` The number of the times a user has been mentioned by other users