AFEL

Analytics for Everyday Learning

EC Project 687916

D1.2 - Base data management infrastructure and core data model

Version 1.1

Deliverable Coordinator: Alessandro Adamou (OU)

Contributors: Mathieu d'Aquin (OU), Besnik Fetahu (LUH)

Reviewers: Esteban Sota (GNOSS), Belgin Mutlu, Peter Hasitschka (KNOW)

This document is part of the AFEL project funded from the European Union's Horizon 2020 research and innovation programme under grant agreement No 687916 - see http://afel-project.org

Change Log

Version	Date	Amended by	Comment
0.1	17/08/2016	A. Adamou	Outline and stub
0.2	08/09/2016	A. Adamou	1st Draft
1.0	13/09/2016	A. Adamou, M. d'Aquin	Version for reviewers
1.1	29/09/2016	A. Adamou, P. Hasitschka, B. Mutlu, E. Sota	Amended per reviewer comments
1.1	29/09/2016	K. Dave	Formatting and finalising report

This document is part of the AFEL project funded from the European Union's Horizon 2020 research and innovation programme under grant agreement No 687916 - see http://afel-project.org

Executive Summary

This is the companion document to the prototype technological stack of the AFEL project, which comprises an online data catalogue (http://data.afel-project.eu/catalogue/), an integrated RESTful Web Service API for submitting and consuming data (documented at http://data.afel-project.eu/api/doc/) and an initial set of pluggable data extraction components, available as open source software and hosted as a group of code repositories on GitHub. The resulting platform accommodates the intuition behind the need for managing different data capture workflows (platform-wide, user-centric), as described in the previous *D1.2 - Specification of data to be collected*. In addition, it elaborates on a third workflow i.e. event-triggered data extraction, which helps control the scale of what content items data needs to be captured from, based on user or system activities.

This document is part of the AFEL project funded from the European Union's Horizon 2020 research and innovation programme under grant agreement No 687916 - see http://afel-project.org

Table of Contents

This document is part of the AFEL project funded from the European Union's Horizon 2020 research and innovation programme under grant agreement No 687916 - see http://afel-project.org

Introduction

This is the companion document to the software system that realises deliverable "D1.2 - Base data management infrastructure and core data model" and that provides a prototypical implementation of the vision emerging from the AFEL data capture specification of D1.1 [AFGYL+16].

The AFEL data platform is a developer resource, therefore it is crucial for it to be accessible through multiple interfaces, namely a user interface (GUI) and a programmatic interface (API) at a very minimum. This document will provide an overview on both interfaces and their underlying principles. For details on both interfaces we will refer to the platform itself, which is self-documenting.

Two classes of extraction components had been identified in D1.1: as their interplay with the data platform differs significantly across such classes, this document will provide a reference to the way the respective workflows from D1.1 are implemented and why the categorisation has been extended to include a third class.

Whilst we expect the basic taxonomy of data sources as in D1.1 to vary only incrementally, and if so required by the evolution of the AFEL data model of everyday learning, dramatic changes are expected in the data sources under these categories. New data sources may arise, and existing sources may upgrade their underlying model, change the update frequency of their data, adopt different policies for data usage, become sporadically unavailable or cease to exist altogether. There are considerations of sustainability and incremental evolution to be made for all such cases. The AFEL data platform is being continuously developed to withstand such changes so that their impact is non-disruptive.

The AFEL data platform exists as a resource for developers within the project team, however it is envisioned that part of it will grow to become an open data hub for learning analysts and application developers in e-learning to use. The potential for growth and evolution of this platform is demonstrated by running deployments of the same infrastructural components in other data-intensive domains such as Smart Cities[1].

[1] An earlier version of the cataloguing and data API infrastructure is part of the MK:Smart DataHub, https://datahub.mksmart.org

This document is part of the AFEL project funded from the European Union's Horizon 2020 research and innovation programme under grant agreement No 687916 - see http://afel-project.org

Design principles

We recap here on the key principles, established at the time of defining the specification of data and their capturing process [AFGYL+16], which directly influence the design and implementation of the core infrastructure, interfaces, data extraction components and the internal data model they work with.

Because of the profound differences in the triggers and rates of generation of relevant data, we need to accommodate *push* and *pull* strategies alike, and establish a mediating framework able to harmonise the two strategies. It was also stated in deliverable D1.1, that the AFEL data infrastructure would operate with decentralised data extraction, with potentially distributed storage mechanisms, yet centralised dataset and data source registries (cataloguing). This ensures a degree of independence, in principle, between data extraction components, but still leaves enough flexibility to design and implement dependencies and cascade extraction processes when the need for them arises.

Extraction components, storage systems, transformation rules and the service interface for receiving data all share the requirement of a common understanding of protocols, formats and representational primitives for data to be exchanged. This requirement implies that there has to be some model in place, even if it were simply a database schema, so that the data can be shaped accordingly by extractors. This data schema is not the same as the model of everyday learning that is being developed as part of the cross-WP work; it will, however, contain the primitives whereupon it is possible to write transformation rules so that data are delivered to client applications in accordance with the model of everyday learning. This decoupling has the advantage that the model of learning can evolve independently. If the model of learning evolves, as expected, in ways so significant as to warrant radical modifications to the low-level data schema, the latter will be extended accordingly. However, having RDF as the underlying representation format allows us the flexibility to perform this kind of refactoring in a monotonic way, so that we can modify the data schema by only *adding* RDF triples..

We can distinguish here between two different types of data refactoring, (i) *soft refactoring*, and (ii) *hard refactoring*[2]. It is expected, that both applications take place in different phases of a workflow by considering the aforementioned design principles .

- **Soft refactoring** is the application of transformation to data according to the way they are exposed to the outside world, without affecting, especially retroactively, the way they are stored.
- **Hard refactoring** is when the raw data need to be transformed in a way that affects the form in which they are stored in the underlying database systems.

[2] This is our terminology that is strictly related to the use cases at hand as as such not present in literature.

This document is part of the AFEL project funded from the European Union's Horizon 2020 research and innovation programme under grant agreement No 687916 - see http://afel-project.org

In order to accommodate the separation between these two types of refactoring, and to incorporate the advantage to sustainability that comes with them, we have envisioned to employ a virtualised form of data integration. Recall from D1.1 that data integration paradigm for AFEL would be characterised by being *mediated*, *pay-as-you-go* and *schemaless* [AFGYL+16]. This means that the collation, aggregation and refactoring of data to be presented through the API will not follow a strictly ETL (Extract-Transform-Load) process, which is typical of data warehousing. Rather, while extraction and loading can be scheduled to be carried out selectively, if at all, and at established intervals, the transformation phase is the core of the whole integration process and is synthesised in a module called mediator. If, due to propagation of requirements of the learning model, the underlying stored data need to undergo hard refactoring (unlike non-extracted data that still reside on their original sources), such an event would be followed by a phase of soft refactoring, where the mappings and transformation rules for the mediator that handle each data source are adapted. This is in effect the pay-as-you-go (PAYG) side of the process, as the price in the analysis and adaptation of transformation rules are paid incrementally as the need for it arises. Schemalessness means, in this case, that the service interface for receiving data needs to accept the data themselves as the payload and not enforce any schema specification. In so doing, we can ensure to be able to ingest data beyond the remit of the core data model and potentially reuse it throughout the PAYG process.

It is important that the application of rules that transform data across models be easy to apply, i.e. without requiring a pass of hard refactoring (or worse, reacquisition) of the source data on every change. This is generally a defining feature of virtual data integration, however, relevant studies propose two distinct ways to transform queries to the integrating party into queries to the respective data sources. In the **global-as-view (GAV)** paradigm the mediator, which resides with the integrating party, processes queries into steps executed at the sources (i.e. the mapping is from the mediated schema to the original sources). In **local-as-view (LAV)** the sources are defined in terms of global relations; the mediator finds all the ways to build a query after having constructed database views (i.e. the mapping is from the original sources to mediated schema) [Cal02]. To draw a quick comparison of the pros and cons of both approaches, LAV requires more sophisticated inferencing than GAV to resolve a query on the mediated schema, thus more complex transformation rules. However, it makes it easier to add new data sources, but this is only given if said schema is stable. Since in AFEL the mediated schema, which reflects our model of learning, is an ever-evolving research product, a purely LAV approach is not completely viable. Still, it is one of the project goals to take advantage of the comparatively stable original sources so that the mappings built for AFEL would be still compatible with the predefined AFEL schema. Therefore, we propose a hybrid approach which:

- inherits from GAV the ability to project (i.e. extract instances) on the local sources, but does not do so with a query (rather, by executing functions in an imperative language);

This document is part of the AFEL project funded from the European Union's Horizon 2020 research and innovation programme under grant agreement No 687916 - see http://afel-project.org

- inherits from LAV the use of bindings to extract a view from the local schemas, but at this stage the data have already been sliced.

Most data extraction components in AFEL will take advantage of the availability of Web Service interfaces from the original data providers or from third-party proxies. It is therefore expected that most, if not all, of the data-related traffic will be through Web services or reengineering of other Web content. As part of making extraction components independent, especially platform-wide ones, it will be possible for extractors to be coupled with Web APIs, as is the case with extractors tied to the Gnoss platform [LSV16]. Specific extraction components, especially user-centric ones, will benefit from the elaboration of data provided by AFEL in addition to those that are specifically provided by that component. For example, as an added value returned to the user for installing that component onto their client systems, an associated application may provide that user information about their learning activities. This calls for a Web Service interface to make the AFEL data as a whole available to developers and users. This service interface will have the following characteristics:

- It will be a RESTful API [RicRub07], with generic Web resources whose URIs are known a priori, and specific resources whose URIs are determined dynamically.
- The specific resources represent relevant entities in the domain of the AFEL learning model.
- It will aggregate data from all the data capture components that are relevant for that specific resources.
- It will require or support client authentication in order to filter out private data that should only be accessible from certain authenticated clients. Credentials for the API are provided to data extractors and end-user applications.
- There will be a set of REST resources with CRUD (Create, Read, Update and Delete) support for datasets obtained by mining data sources. Client authentication will be enforced on these resources.
- It will provide public but potentially sensible data in anonymised form.
- It will incorporate support for the discovery of the URIs of specific REST resources, still within the restrictions enforced by client authentication.

The last high-level requirement is a consequence of having to consider the variability of data items on sources that are beyond the control of the project. Consequently the REST API of the data platform must be designed to take into account discoverability of its resources, because these may change as the original data do, and it would be unthinkable to document an extremely vast, if finite, set of REST resources one by one.

This document is part of the AFEL project funded from the European Union's Horizon 2020 research and innovation programme under grant agreement No 687916 - see http://afel-project.org

Technology overview

What follows is a summary of the grounding for the technology stack of the initial AFEL data platform and is expected to drive its evolution over the course of the project.

Data cataloguing. In the light of the uptake of Open Government Data, there has been a significant technological focus, partially backed by research, on how to provide centralised registries of distributed datasets around a specific domain [KCN16]. Catalogues can be employed either as supporting registries in the literal sense, in that external access points to the datasets are provided but control is deferred to them, or supporting data repositories, where a degree of access to and consumption of the data is provided by the catalogue itself. The most popular open source software system to support the former is *CKAN* [Win16], which reflects onto data management the basic notions behind software package managers. Software platforms employed to support data repositories include the open source suite *Dataverse*[3] and the commercial platform *Socrata*[4]. Whilst all of these systems sport a significant number of adoption instances, such as DataHub.io, their general-purpose nature and the need to accommodate datasets whose specifications are very volatile and variable in granularity is still causing many use cases of data cataloguing to be implemented in-house using solutions developed ad hoc. For instances, the need to formally represent policies on data and to reason on how they propagate across datasets was part of the motivation for bespoke data cataloguing technologies to be developed in the Smart Cities domain as in the MK:Smart datahub[5] [DAdA16].

Linked Data. One of the leading paradigms for publishing on the Web machine-readable data that lend themselves to processing by machine learning algorithms is *Linked Data*. This is a collection of principles for identifying and referencing entities as resources on the Web, and proposed standards for representing and querying the resulting data, namely the RDF format and SPARQL query language [BHBL09]. The principles behind Linked Data are adopted throughout the design of the AFEL data platform, however, in order to abate the cognitive overhead for developers to learn RDF and SPARQL, we set out to provide ad-hoc query languages and representation formats for integrated data, however the possibility to also publish AFEL data according to the recommended standards is being considered for the consolidated version of the data platform. Internally, however, RDF and SPARQL are fully supported as, respectively, the format in which extracted data are stored and the query language to which integration rules are transformed in order to fetch data from external sources. A battery of RDF-based quad stores and SPARQL engines, using the popular *Apache Jena Fuseki* server[6], is deployed in the back-end of the AFEL data platform. While

[3] Dataverse, http://dataverse.org
[4] Socrata, https://socrata.com
[5] MK:Smart Datahub, http://datahub.mksmart.org
[6] Apache Jena, http://jena.apache.org

This document is part of the AFEL project funded from the European Union's Horizon 2020 research and innovation programme under grant agreement No 687916 - see http://afel-project.org

we envision the development of data integration rules and libraries that directly support other formats without necessarily going through RDF and SPARQL, this will still be the default formalism to be adopted whenever data need to be reengineered and stored as assets of the project. In that light, part of future work being considered is to implement strategies for optimising data consumption that either mimic of fully support the recent developments in Linked Data Fragments [VSCCMW14]. It is an interface for consuming and serving Linked Data across certain patterns that equally distribute the load across client and server systems. Support for Linked Data Fragments is expected to become increasingly useful as extractors and data-intensive end user apps are developed by the AFEL project.

SQL-like and NoSQL databases. The aforementioned triple and quad stores are examples of databases whose status in the SQL/NoSQL dichotomy [Str16] is disputed, because of the way the SPARQL 1.1 query language has evolved to support a more SQL-like syntax[7]. However, actual databases that employ different paradigms other than the SQL-like one have a margin of usefulness for efficiently handling intermediate operations that have highly different scale requirements with each other, depending on either the size of the data or their update frequency. Therefore they may on occasion be employed as part of specific phases of the data management workflow. Among these we cite **document databases** such as *Elastic*[8] and *CouchDB*[9] (employed by the main data API for real-time management of integration rules and data caching); **graph stores** such as *Neo4j*[10] and *AllegroGraph*[11], and **wide column stores** such as *Hadoop/HBase*[12] and *Cassandra*[13].

RESTful Web Services. The REpresentational State Transfer paradigm has proven particularly apt for the formulation and implementation of Web Service APIs where every actionable unit is a Web resource whose state is acted upon through standard HTTP [RicRub07]. The popularity of this paradigm has given rise to several native client libraries and server frameworks for providing and invoking such services. These are available in several languages such as Java (*JAX-RS*[14]), JavaScript/Node.js (*Gugamarket*[15]), PHP (*Zend*[16]), Python, Rails and C# (.NET Framework). The RESTful model is the basis for all the APIs provided by the AFEL project, and their implementations are carried out by combining the aforementioned libraries and frameworks as needed by the respective languages.

[7] Despite being essentially graph-based, SPARQL-based implementation are not considered part of NoSQL canon, http://nosql-database.org
[8] Elastic Search, http://www.elasticsearch.org
[9] Apache CouchDB, http://couchdb.apache.org
[10] Neo4j, http://www.neo4j.org
[11] AllegroGraph, http://www.franz.com/agraph/
[12] Hadoop/HBase, http://hadoop.apache.org
[13] Apache Cassandra, https://cassandra.apache.org
[14] JAX-RS specification, https://jax-rs-spec.java.net
[15] Gugamarket, http://www.gugamarket.com
[16] Zend Framework, http://framework.zend.com

This document is part of the AFEL project funded from the European Union's Horizon 2020 research and innovation programme under grant agreement No 687916 - see http://afel-project.org

The AFEL data platform

"AFEL data platform" is an umbrella term that encompasses the set of developer resources for the project team to coordinate their efforts of extraction and management for all the data which may be useful in other tasks of the project. Since this initial version, it is being built with a focus on evolution, so as to become a set of resources for all developers and specialists in technology-enhanced learning to use and build their applications upon.

The AFEL data platform is entirely web-based and contains the following resources:

- A human-readable dataset catalogue at http://data.afel-project.eu/catalogue/.
- A RESTful Web service interface for programmatic access to the data, available at http://data.afel-project.eu/api/ and documented at http://data.afel-project.eu/api/doc/.
- A set of data extraction components, distributed across the Web and, in some cases, runnable on the client systems, all available as open source software.

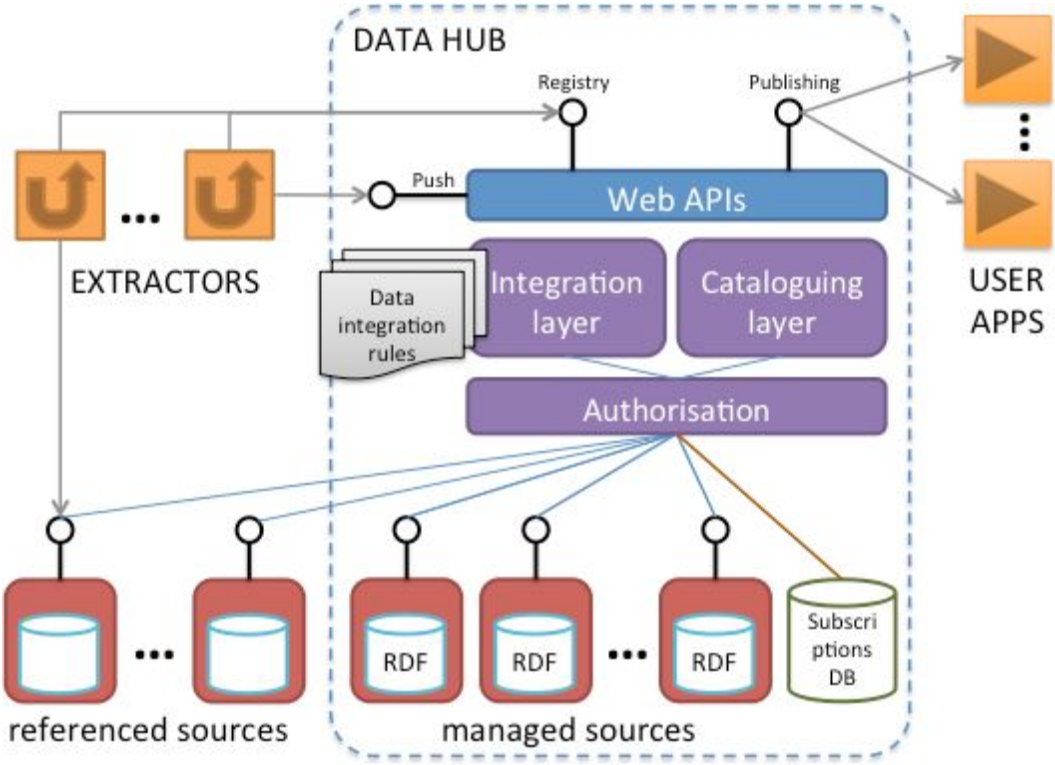

Figure 1: Architecture of the AFEL data platform.

The resources mentioned above are the public-facing facade of the entire management framework whose architecture is summarised in Figure 1. The underlying logic predicates that the data management workflow must be able to operate in a distributed way. The *data hub* is the set of core software components that interoperate with data sources and data consumers alike. It consists of (a) a set of *managed* data sources (i.e. those whose entire life-cycle is controlled by the core platform) based on RDF stores; (b) a *cataloguing layer* for

This document is part of the AFEL project funded from the European Union's Horizon 2020 research and innovation programme under grant agreement No 687916 - see http://afel-project.org

registering new datasets produced by running extractors on data sources; (c) an *integration layer* which aggregates and assembles data from all the datasets registered with the catalogue; and (d) Web APIs for accessing or pushing data, as well as interacting with the cataloguing layer programmatically. It should be noted here that these components need not be co-located in a strictly centralised way: the triple stores in the back-end of some managed sources are indeed distributed. *Extractors* are independent software components responsible for referencing external data sources and capturing their content in the form of datasets. Extractors may independently interact with their own storage systems (as part of *referenced* data sources) or directly push the data to the data hub and request it to create managed data sources: in either case they are required to register the metadata of the corresponding source with the catalogue, so that it is possible to define data integration rules which include referenced sources in the process. Cataloguing, receiving and exposing data are all operations which are tightly bound to an *authorisation* functionality in the data hub. This layer is essentially responsible for generating API keys for users and applications, controlling access permissions to the datasets based on the API keys provided in Web Service calls and managing user subscriptions to the datasets they are allowed to access.

In order to provide added value to the users who agree to have their data privately submitted to the AFEL platform, possibly installing extraction components on their systems, it is possible to develop end-user applications that make use of the data publishing API. The development of some such applications is planned as part of the project (see deliverable D3.1), especially in conjunction with specific user-centric data extractors, hence their inclusion in Figure 1. However, other applications may be developed by third parties once the AFEL data platform becomes an open developer toolkit and cleared of sensitive user data.

The data platform is based on the principle that regulates data governance in AFEL and can be summarised as "*datasets in, entities out*". This paradigm essentially states that the methods by which data are served back to clients and developers transcend the boundaries of the dataset where they originally belong. In other words, users and client applications will not issue a request to view a dataset, but to view all the data that all the datasets can provide about a specific entity in the domain of everyday learning. This is the leading principle of the data integration effort which is carried out as part of the data life-cycle, and is the main reason why existing off-the-shelf data cataloguing tools, whether they provided direct data access from within the catalogue (e.g. Socrata) or not (e.g. CKAN), were not deemed suitable for the controlled data management environment that AFEL strives to provide.

The source code of the data cataloguing software, APIs and extractors is freely available under the GitHub organisation `afel-project` at https://github.com/afel-project.

Data management interfaces

When a dataset is registered with the AFEL data platform, either by a contributing user or automatically by an extraction component, the metadata regarding the sources (e.g.

This document is part of the AFEL project funded from the European Union's Horizon 2020 research and innovation programme under grant agreement No 687916 - see http://afel-project.org

provenance, license) and the content itself (e.g. the corresponding categories as per AFEL deliverable 1.1 [AFGYL+16]) can also be added to the AFEL catalogue. These metadata can then be accessed either programmatically via client applications, or via a user interface. The data themselves are only meant to be accessed programmatically, although facilities for previewing and exploring them are in place as part of the catalogue GUI (cf. Figure 3).

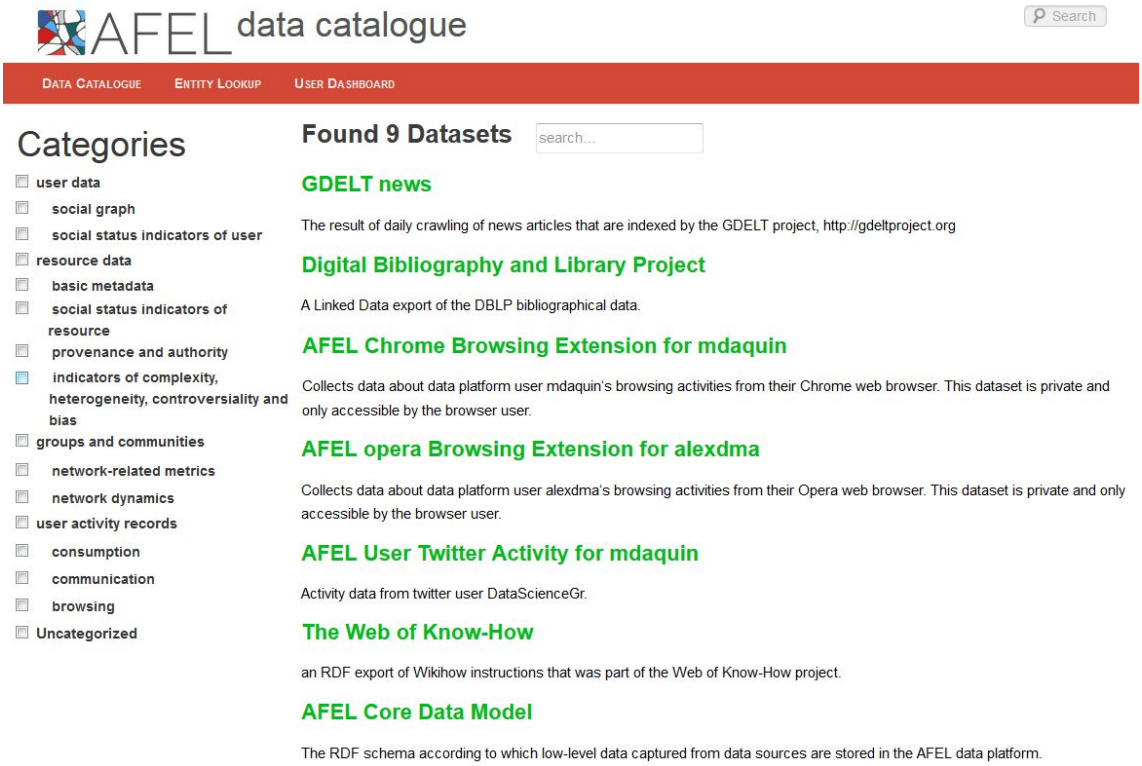

Figure 2: Screenshot of the prototypical GUI of the AFEL data catalogue.

The metamodel of the catalogue assumes that a dataset is the largest unit so that an access control policy can be applied atomically to it and spans uniformly throughout its content. Thus, a single source may generate multiple datasets: an example is given by data extraction components (cf. section "User-centric extractors"), where each instance of the component, as run by an external user of the system where the component is installed, can create its own dataset whose data are sensitive and therefore require stringent access policies. Such a feature, where the customisation of dataset granularity is somewhat implicit and not directly customisable, can hardly be found in existing data catalogue software [DAdA16].

APIs

The AFEL data platform provides programming interfaces (APIs) which allow applications to consume, provide and browse its data. Thus, it is possible to actually explore and discover data types and their instances without having to go through the catalogue user interface at all. These APIs are self-documented, in that dynamic, interactive online documentation is available alongside the Web Service endpoint, and this documentation allows developers to

This document is part of the AFEL project funded from the European Union's Horizon 2020 research and innovation programme under grant agreement No 687916 - see http://afel-project.org

test service calls directly on the endpoint itself[17]. Because of this feature and the prototypical hence evolving nature of the APIs, only the salient REST resources and operations thereon will be reported here, whilst we refer to the online documentation for the full specification.

The APIs for providing and consuming AFEL data effectively realise the "datasets in, entities out" paradigm by providing explicit REST resources for both classes of artefacts, and implementing total separation of competences between them and the operations. In other words, write operations (create, update, delete) are only possible for datasets, and read operations on datasets and entities yield orthogonal results. JSON is the format of choice for all the response bodies of these APIs.

Figure 3 is a snapshot of the online documentation of the APIs for dataset management and entity consumption. The Web Service interface is a RESTful API in that it provides the following features:

- Identification of resources (datasets or entities in the informal learning domain) through URIs, all starting with the base URI http://data.afel-project.eu/api/[18].
- Standard HTTP protocol methods and response codes to reflect the way an operation affects the state of a resource.
- Separation of resources from their representations, though at present only one JSON serialisation is available.

[17] Recall that the interactive API documentation is available at http://data.afel-project.eu/api/doc/.
[18] Subject to change, as the `/api` element might be deprecated in the future.

This document is part of the AFEL project funded from the European Union's Horizon 2020 research and innovation programme under grant agreement No 687916 - see http://afel-project.org

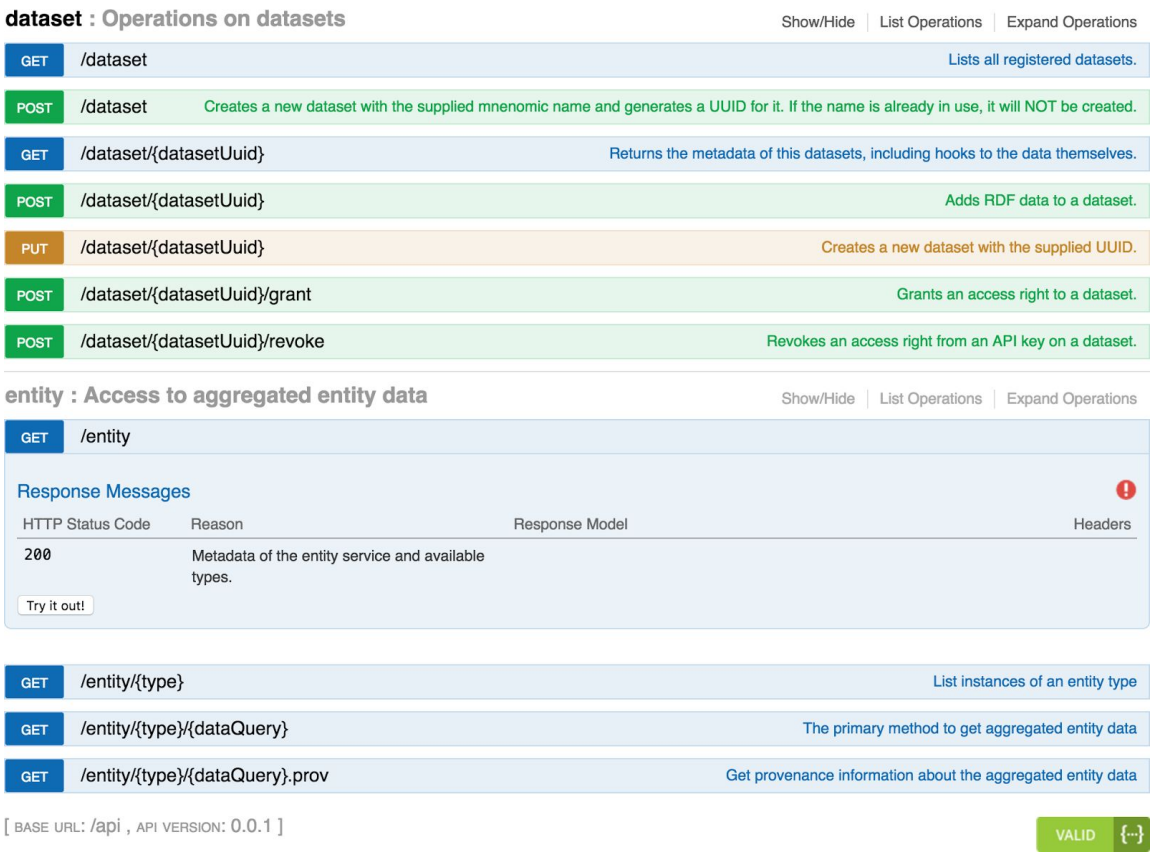

Figure 3: Online specification of the AFEL data API created using *Swagger UI*[19].

As shown in Figure 3, there are two distinct paths that denote two separate classes of REST resources:

- `/dataset` is the path prefix of all the resources which allow a dataset to be created, deleted, have data added to it and have its metadata inspected.
- `/entity` is the path prefix of all the resources that allow the data present in all the datasets on the platform to be viewed in terms of the entities they are about; it incorporates, in effect, a very simple query language.

Sub-resources of `dataset` are named by a unique identifier that can be either supplied via a direct PUT request (if valid and not already taken), or by letting the API chose via a POST request to the super-resource.

The `entity` endpoint is the one through which the actual data aggregated from the datasets registered with the AFEL catalogue are exposed. An entity is any semantically relevant unit of knowledge in the domain of learning, such as a user, learning resource, system, social network, topic, location or time period. Sub-resources of `entity` are the actual semantics-laden units of knowledge that a client may want to obtain all the allowed data about. The `dataQuery` placeholder in the specification of Figure 2 currently supports an

[19] Swagger UI, http://swagger.io

This document is part of the AFEL project funded from the European Union's Horizon 2020 research and innovation programme under grant agreement No 687916 - see http://afel-project.org

extremely simple identifier-based query mechanism that translates to "*give me all the data that I can access concerning the entity identified by* dataQuery". Some of the possible identifiers to be used as values for this placeholder can be discovered by performing a GET request on the dataset super-resource. More complex queries will be supported in future versions of the API, effectively developing a flexible ad-hoc query language. The output of a service call to an entity resource is a representation of all the aggregated data about that entity that the client is allowed to access (as determined by the supplied API key). It is also possible to obtain provenance information by querying a derivative resource of the one being queried for data. This is done by appending the .prov suffix to the original resource, which for each property path in the data returns the list of datasets that contributed to delivering values.

Note that, in conformance with the "datasets in, entities out" paradigm, sub-resources of entity only accept GET requests, meaning that direct modifications to the data are not allowed by performing write operations on their representations. The appropriate way to modify data is by a POST request containing the new data to the dataset resource that needs modifying.

Most resources support HTTP authorisation as a way to control which datasets can be managed via the dataset endpoint and which ones are considered when returning data about an entity via the entity endpoint. Authorisation tokens are handled based on the provision of an API key alongside the request. This API key is mandatory for all requests that use the POST, PUT or DELETE methods. However, although it is optional for GET requests, not providing an API key may cause an empty JSON response to be delivered, if there are no *open datasets* that contribute to fulfilling the data request. At present, only authorisation via HTTP query parameter or Basic Authentication [FieRes14] is supported: ways to support other authorisation methods such as *OAuth*[20] are currently being investigated; in particular, support for OAuth is expected to be introduced as user-centric extractors for social networks are developed, which is expected to take place early in the second year of the project.

Initial extraction components

Deliverable D1.1 described a set of data sources requiring two different types of extraction components, in order for these sources to become part of the AFEL data platform: (a) *platform-wide extractors*, which crawl existing online platforms to get data about, for example, the resources they contain; and (b) *user-centric components* that obtain data directly from user applications regarding their online activities. Through designing the data platform and identifying common scenarios, we introduced a new notion of an "*event-triggered*" extractor component, which sits in between a user-centric component and a platform-wide component. These extractors obtain data about a specific resource, or set thereof, within a

[20] A protocol for cross-site secure authorisation, https://oauth.net.

This document is part of the AFEL project funded from the European Union's Horizon 2020 research and innovation programme under grant agreement No 687916 - see http://afel-project.org

given platform at the time they are required according to parameters set by a user-centric extractor.

Further details about the three types of extractor components and their developing process within the project are given below.

Platform-wide extractors

Platform-wide extraction components represent the simplest form of components conceptually. They are mostly crawlers and back-end scripts that use existing APIs or data access to platforms, such as educational platforms, to obtain batch data about the resources, user activities on these platforms. These extractors might require to be developed in different ways, as long as they fulfill the following requirements:

- the resulting dataset should be in RDF;
- the resulting dataset should be catalogued in the AFEL Data Catalogue;
- the RDF data should be transferred into the data platform using the data platform's API with a key providing access to the platform.

This category of extractors includes the components that enable the transfer of data from the GNOSS platform, based on the technical components described in AFEL deliverable D5.2 [LSV16].

Other extractors are currently also at different stages of development which mostly obtain data from resource providers. Those include the *GDelt* crawler[21] which crawls news stories obtained from the GDELT index[22]. Other components being developed in this category include crawlers for *SlideShare*, *bibsonomy*, *DBLP* and *Flickr*.

Finally, there is a range of entity-centric extractors that select facts about entities from Web crawls like the *Web Data Commons*[23]. In addition to fact selection, other components take this further and use the selected facts for augmenting knowledge bases with facts about an entity. In the case of AFEL, the union of all the datasets generated firsthand by data extractors of the project, combined with the external datasets registered with the data platform, would constitute the corresponding "AFEL Knowledge Base".

User-centric extractors

User-centric extraction components send data to the platform which relate to a specific user. In many cases, theses user-centric extractors relate to user activities and will generate data, for example, when the user consumes a resource or carries out a specific action.

[21] GDelt crawler source code at https://github.com/afel-project/gdelt_crawler
[22] GDELT, http://gdeltproject.org
[23] Web Data Commons, extracting structured data from the Common Crawl, http://webdatacommons.org

This document is part of the AFEL project funded from the European Union's Horizon 2020 research and innovation programme under grant agreement No 687916 - see http://afel-project.org

Typically, their interaction with the data platform is more complex than for platform-wide extractors. Concretely, user-centric extractors operate as follows:

- they are user-facing and collect personal data;
- they send data as a stream and not as a batch;
- the data being sent should only be writable and readable by the user from which it originates.

This interaction is illustrated in Figure 4 below. The figure shows the workflow from the time the user either installs the extractor (on their local machine or on an online platform), or otherwise activates the corresponding functionality by accepting its terms and conditions (if the extractor is already provided as an optional functionality of the online platform).

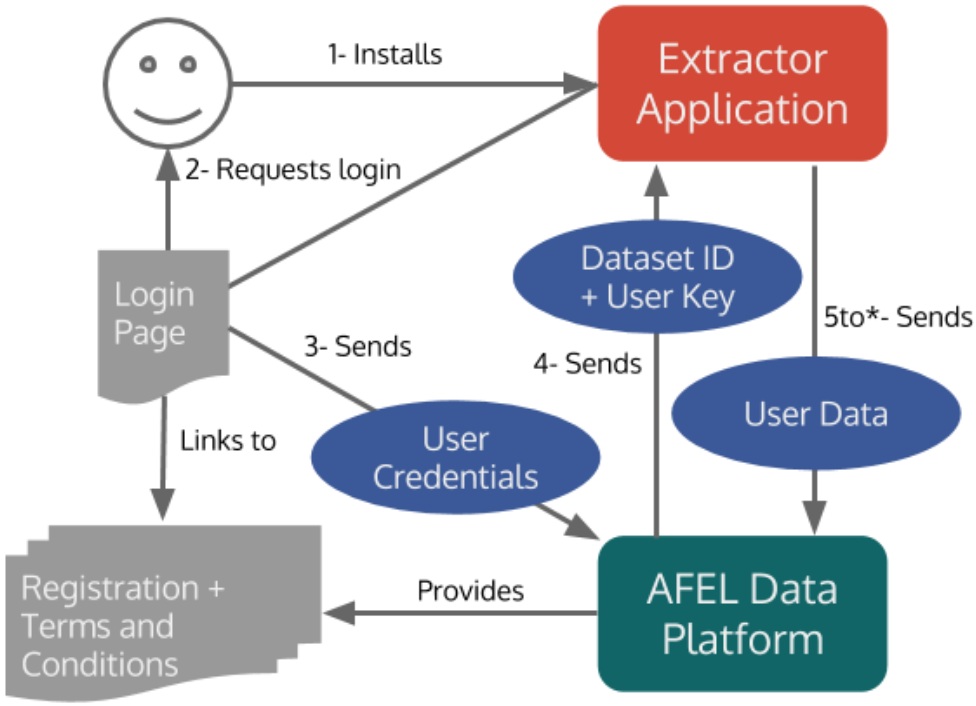

Figure 4: Workflow of a default user-centric data extractor.

Regarding the workflow, the user's first action after the installation will be to log into the AFEL data catalogue using a dedicated login form provided by the extractor application. This is necessary for the AFEL data platform to create a dataset on the catalogue and set the credentials so that only this user will be able to access the dataset via the data API of the platform. It is also worth noting that this is where the user can be exposed to the Terms and Conditions of the extractor, where in particular the policy regarding the data being transmitted is described. The catalogue returns to the application an identifier for the newly created dataset, as well as the unique user key that enables data transfer to this dataset. It is

This document is part of the AFEL project funded from the European Union's Horizon 2020 research and innovation programme under grant agreement No 687916 - see http://afel-project.org

also recommended that the application provides a link to the user dashboard on the AFEL data platform (see deliverable D3.1).

A number of extractors are currently being implemented in this category, again here at different stages of development. The Browsing History extractor[24], for example, implements the whole workflow described above and is currently operational, as it is presently generating data from specific members of the project acting as testers. This extractor, which is packaged as a *WebExtension* to work on the latest Google Chrome, Opera and Firefox, sends basic activity data about the Web pages visited by the user through the browser (including URL and title of the page, as well as the time at which the page was visited). This extractor was developed to represent a template of the workflow described above for others to reproduce and clearly emphasises the need for clear policies regarding the handling of user activity data (deliverable D6.3, forthcoming).

Following this example, other extractors are currently being developed, including a similar extension as the one described above for the Firefox browser. A simple application (see code) also applies this process to register the user's activity on Twitter (based on an RDF twitter extractor component). We are also creating extractors that can take user activity data from common social media platform such as Facebook and Instagram.

Event-triggered extractors

Through conceptualising scenarios involving user-centric extractors, we identified the need for a new type of extractors. Indeed, an extractor such as the browsing history one described above will include activities that relate to the consumption of specific resources - a *YouTube* video or a post on *StackOverflow* - but will not include any information about this resource. The idea here is to include a mechanism in the data platform through which the events such as "data about the consumption of a resource from {a specific platform} by {a specific user} have been uploaded", thereby triggering a dedicated extractor whose function is to obtain information about this specific resource.

Since the need for this kind of extractors was only identified recently, the corresponding mechanism will only be available in the next release of the AFEL data platform (deliverable D1.4, forthcoming). However, a number of this kind of extractors are already being planned for platforms such as *YouTube*, *SlideShare*, *StackOverflow* and *LinkedIn*.

Core data model

As described above, as well as in deliverable D1.1, data ingestion in AFEL is *schemaless*, in that there is no requirement for the different data sources to adhere to the same schema, or in fact to any predefined schema. However, there are two reasons for the project to create a data model as a core reference model for all partners: (1) The API is configured to output JSON

[24] AFEL Browser History WebExtension, https://github.com/afel-project/browsing-history-webext

This document is part of the AFEL project funded from the European Union's Horizon 2020 research and innovation programme under grant agreement No 687916 - see http://afel-project.org

data which, if configured by different members of the project for different data sources, need to be consistent, and (2) a data model provides guidelines regarding the type of data and the attributes of data entities we are expecting to manipulate through different data sources.

An initial data model/schema has been created, on the basis of the taxonomy of data sources described in D1.1. The main design choice here was to rely on schema.org as the basis for this schema, and to extend it with elements that are not already present. Indeed, it naturally includes concepts and properties to describe user profiles and resources, and through the LRMI initiative, already has extensions to tackle learning resources specifically. Some aspects where extensions are required however include entities and attributes related to the user's activities, where a small vocabulary of actions exist, but is not always sufficient.

The core data model is currently available in RDF through the AFEL Data Platform's catalogue at http://data.afel-project.eu/catalogue/dataset/afel-core-data-model/.

This document is part of the AFEL project funded from the European Union's Horizon 2020 research and innovation programme under grant agreement No 687916 - see http://afel-project.org

Conclusion

The initial version of the AFEL data platform and the underlying model of data managed by it provides prototypical implementations of the entire principal technology stack of data capture. This deployment effectively realises the full data capture cycle for a bootstrap set of data sources of several types, according to both the data source taxonomy from D1.1 and the workflows that need to be realised for data capture to be focussed and scalable. In the spirit of distributed data repository management, this framework allows the flexibility for different data stores to be used, provided an update service that supports them is installed.

Particularly, the AFEL data platform as a whole already provides facilities: (a) for extraction components to push data to general-purpose storage or to dedicated storage, on a regular basis if needed; (b) for extraction components to set the access permissions on datasets for specific users; (c) for data consumer applications to access the data they require through a dedicated API that does not require indications of the datasets to use; and (d) for server-side extractors to be triggered upon certain data-related events.

While the prototypical data platform provides the basic features to enable the creation of data extractors, future work throughout the project will focus on leveraging their versatility so that other instantiations of the same use cases can be accommodated. Because the development of new data extractors as indicated in the D1.1 plan will continue, so too will specific requirements emerge with respect to e.g. the parts of the core data schema to be used, formats to be supported and other scale factors for push and pull strategies to be considered. The ongoing evolution of the AFEL data platform concentrates not only on supporting these variants, but also on making certain processes easier, if not automated. For example, when a new data source is included in the capture process, it will be possible to ease the definition of data integration rules for user applications, by inheriting them from the rules of datasets that are similar in their representation or semantics.

This document is part of the AFEL project funded from the European Union's Horizon 2020 research and innovation programme under grant agreement No 687916 - see http://afel-project.org

References

[AFGYL+16] Adamou, Alessandro, Besnik Fetahu, Ujwal Gadiraju, Ran Yu, Elisabeth Lex, Matthias Traub, Peter Holtz, Ricardo A. Maturana, Esteban Sota, Susana López-Sola, and Mathieu d'Aquin. *D1.1 - Specification of data to be collected*. AFEL project deliverable (2016).

[BHBL09] Bizer, Christian, Tom Heath, and Tim Berners-Lee. Linked Data - The story so far. *Int. J. Semantic Web Inf. Syst.* 5:3 (2009): 1-22.

[Cal02] Calì, Andrea. Reasoning in data integration systems: Why LAV and GAV are siblings. *Foundations of Intelligent Systems* (2003): 562–571.

[DAdA16] Daga, Enrico, Alessandro Adamou, and Mathieu d'Aquin. Addressing exploitability of Smart City data. *IEEE International Smart Cities Conference* (2016).

[FieRes14] Fielding, Roy and Julian Reschke (eds.). Hypertext Transfer Protocol (HTTP/1.1): Authentication. IETF RFC 7235 (2014). https://tools.ietf.org/html/rfc7235

[KCN16] Kucera, Jan, Dusan Chlapek, and Martin Necasky. Open Government Data Catalogs: Current Approaches and Quality Perspective. In *Second Joint International Conference on Electronic Government and the Information Systems Perspective, and Electronic Democracy, EGOVIS/EDEM* (2013): 152-166.

[LMSGD+16] López-Sola, Susana, Ricardo A. Maturana, Esteban Sota, Ujwal Gadiraju, Stefan Dietze, Mathieu d'Aquin, and Alessandro Adamou. *D5.1 - Description of the available social environments and data*. AFEL project deliverable (2016).

[LSV16] López-Sola, Susana, Esteban Sota, and Juan Valer. *D5.2 - GNOSS-Didactalia base data extractors for the AFEL data platform, release 1*. AFEL project deliverable (2016).

[RicRub07] Richardson, Leonard and Sam Ruby. RESTful web services - web services for the real world (2007).

[Str16] Strauch, Christoph. NoSQL whitepaper. *Stuttgart: Hochschule der Medien* (2012). http://www.christof-strauch.de/nosqldbs.pdf

[VSCCMW14] Verborgh, Ruben, Miel vander Sande, Pieter Colpaert, Sam Coppens, Erik Mannens, and Rik van de Walle. Web-Scale Querying through Linked Data Fragments. In *Proceedings of the Workshop on Linked Data on the Web co-located with the 23rd International World Wide Web Conference (WWW 2014)* (2014).

This document is part of the AFEL project funded from the European Union's Horizon 2020 research and innovation programme under grant agreement No 687916 - see http://afel-project.org

[Win16] Winn, Joss. Research Data Management using CKAN: a Datastore, Data Repository and Data Catalogue. In *IASSIST 2013 - Data Innovation: Increasing Accessibility, Visibility, and Sustainability* (2013).

This document is part of the AFEL project funded from the European Union's Horizon 2020 research and innovation programme under grant agreement No 687916 - see http://afel-project.org