D2.4 Software prototype v1

DURAARK

FP7 – ICT – Digital Preservation
Grant agreement No.: 600908

Date: 2014-07-31
Version 1.0
Document id. : duraark/2014//D.2.4/v1.0

Grant agreement number	:	600908
Project acronym	:	DURAARK
Project full title	:	Durable Architectural Knowledge
Project's website	:	www.duraark.eu
Partners	:	LUH – Gottfried Wilhelm Leibniz Universitaet Hannover (Coordinator) [DE] UBO – Rheinische Friedrich-Wilhelms-Universitaet Bonn [DE] FhA – Fraunhofer Austria Research GmbH [AT] TUE – Technische Universiteit Eindhoven [NL] CITA – Kunstakademiets Arkitektskole [DK] LTU – Lulea Tekniska Universitet [SE] Catenda – Catenda AS [NO]
Project instrument	:	EU FP7 Collaborative Project
Project thematic priority	:	Information and Communication Technologies (ICT) Digital Preservation
Project start date	:	2013-02-01
Project duration	:	36 months
Document number	:	duraark/2014/D.2.4
Title of document	:	Software prototype v1
Deliverable type	:	Software prototype
Contractual date of delivery	:	2014-07-31
Actual date of delivery	:	2014-07-31
Lead beneficiary	:	Fraunhofer Austria (FhA)
Author(s)	:	Martin Hecher <martin.hecher@fraunhofer.at> (FhA) Dag Field Edvardsen <dag.fjeld.edvardsen@catenda.no> (Catenda) Sebastian Ochmann <ochmann@cs.uni-bonn.de> (UBO) Michael Panitz <michael.panitz@tib.uni-hannover.de> (LUH) Hamid Rofoogaran <hamid.rofoogaran@ltu.se> (LTU) Ujwal Gadiraju <gadiraju@l3s.de> (L3S) Besnik Fetahu <fetahu@l3s.de> (L3S)
Responsible editor(s)	:	Martin Hecher <martin.hecher@fraunhofer.at> (FhA)
Quality assessor(s)	:	Jakob Beetz <j.beetz@tue.nl> (TUE) Martin Tamke <martin.tamke@kadk.dk> (CITA)
Approval of this deliverable	:	Jakob Beetz <j.beetz@tue.nl> (TUE) Stefan Dietze <dietze@l3s.de> (LUH)
Distribution	:	Public
Keywords list	:	prototype, workbench, use cases

Executive Summary

This report describes the first version of the integrated software prototype comprising the software prototypes developed in DURAARK so far. It exposes the functionality of the prototypes as a service-oriented platform (the "Workbench") and provides it to stakeholders via a coherent graphical user interface (the "WorkbenchUI"), yielding an integrated application for performing long-term archival tasks for BIM data from the view of a front-end stakeholder. Additionally, the software acts as a service provider for third party developers to be able to integrate the functionality developed in DURAARK in their own (existing) applications.

The report guides a stakeholder through the usage of the graphical user interface, describes the components on a technical level and gives interested readers and developers information on how to use the Workbench as a service provider.

Table of Contents

1 Introduction

This report describes the first version of the integrated software prototype, which is referred to as the ***DURAARK Workbench*** in the remainder of the document. The purpose of the Workbench is to provide an integrated platform for the software deliverables developed in the project, as well as the future ones. Currently the following software prototypes are included:

- The **Workbench** acting as service-oriented platform for the functionality developed in DURAARK and providing a coherent web-based user interface to access the functionality from a stakeholder point of view. The user manual and the technical architecture description are available in this report.
- The **Semantic Digital Archive (SDA)** which consists of a number of sub-components integrated into the workbench. While their general use is described in this report, more in-depth technical aspects can be found in report **D3.3** describing the first SDA prototype.
- The **Point Cloud tools** responsible for the geometric enrichment of E57 files. Those tools are generating additional files containing corresponding information which is then uploaded to the preservation system via the Workbench. From the set of planned tools this version of the integrated prototype contains the *point cloud registration prototype* described in report **D4.1**. There is no integration of the software deliverable produced in **D5.1**, yet. The software is responsible for recognition of meaningful shapes and point cloud compression. The integration will be done for the milestone in M30, which will also contain **D5.2**, due in M20. This way the M30 prototype will contain both WP5 deliverables in a consistent way (from the view of a stakeholder) to extend the Workbench with WP5's topic "Recognition of Architecturally Meaningful Structures and Shapes". Also, the point cloud compression feature will only be used in M30 for providing the stakeholder with a interactive 3D preview for a point cloud. From an implementation point of view the integration of D5.1 and D5.2 are very similar to D4.1, there are no conceptual tasks left to solve for their integration.

Figure 1 gives an overview of the structure of the reports that accompany this software deliverable.

With the integrated software prototype a stakeholder is able to perform a selection of use cases defined in report **D2.1**. The selection is the following:

- **UC1:** Deposit 3D architectural objects
- **UC2:** Search and retrieve archived objects
- **UC3:** Maintain Semantic Digital Archive
- **UC8:** Exploit contextual information for urban planning
- **UC9:** Enrich BIM/IFC model with metadata from a repository

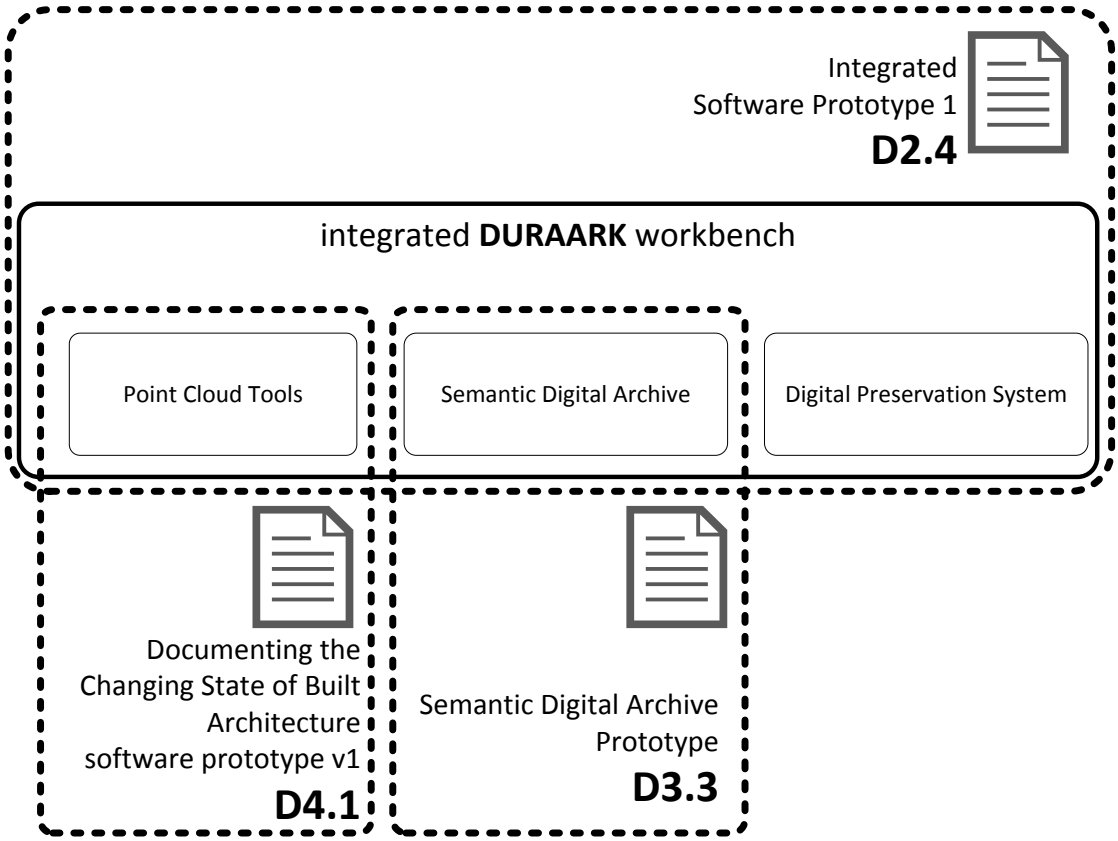

Figure 1: **Overview of the scope of the M18 software prototypes and respective reports**

The workbench organizes the use cases into ***workflows***. A workflow is a step-by-step process on how to achieve the purpose of one or multiple use cases. For instance, one of the implemented workflows handles the generation of a SIP (Submission Information Package) file from a set of given input files. In this case the workflow covers UC1, the deposition of 3D architectural objects, as well as UC9, the enrichment of the BIM/IFC model with metadata from a repository. This is a list of the workflows provided by the workbench so far:

SIP Generation A stakeholder selects a set of input files describing a building. After a file identification the automatically extracted metadata of the files is shown and editable. Based on the metadata an automatic enrichment with *Linked Open Data* is performed and stored in a metadata record. In a final step, the input files and the metadata record are archived into a downloadable SIP file and the metadata record of the SIP is indexed into a PROBADO3D database for later search & retrieval.
Covered use cases: UC1, UC8

Search & Retrieval A stakeholder is provided with a list of generated SIP files. Metadata records for the SIP can be displayed. A full-text search within all metadata records allows the stakeholder to filter the list of files.
Covered use case: UC2

Geometric Enrichment The geometric enrichment workflow is based on the desktop application yielded from the **software deliverables D4.1 and D5.1** in M12. After the selection of one or multiple IFC and E57 files a stakeholder is provided with a graphical user interface for performing a geometric registration of the input files. The process yields a mapping file that can be added to a SIP file in within the **SIP Generation** workflow.
Covered use case: UC9

Semantic Archive Maintenance The maintenance of the SDA component is managed by this workflow. A stakeholder is provided with graphical user interfaces for i) the content of the SDO and SDA sub-components that include crawling, profiling and archiving evloving temporal states of the Linked Datasets used in the Long Term Preservation scenarios covered by DURAARK.
Covered use case: UC3

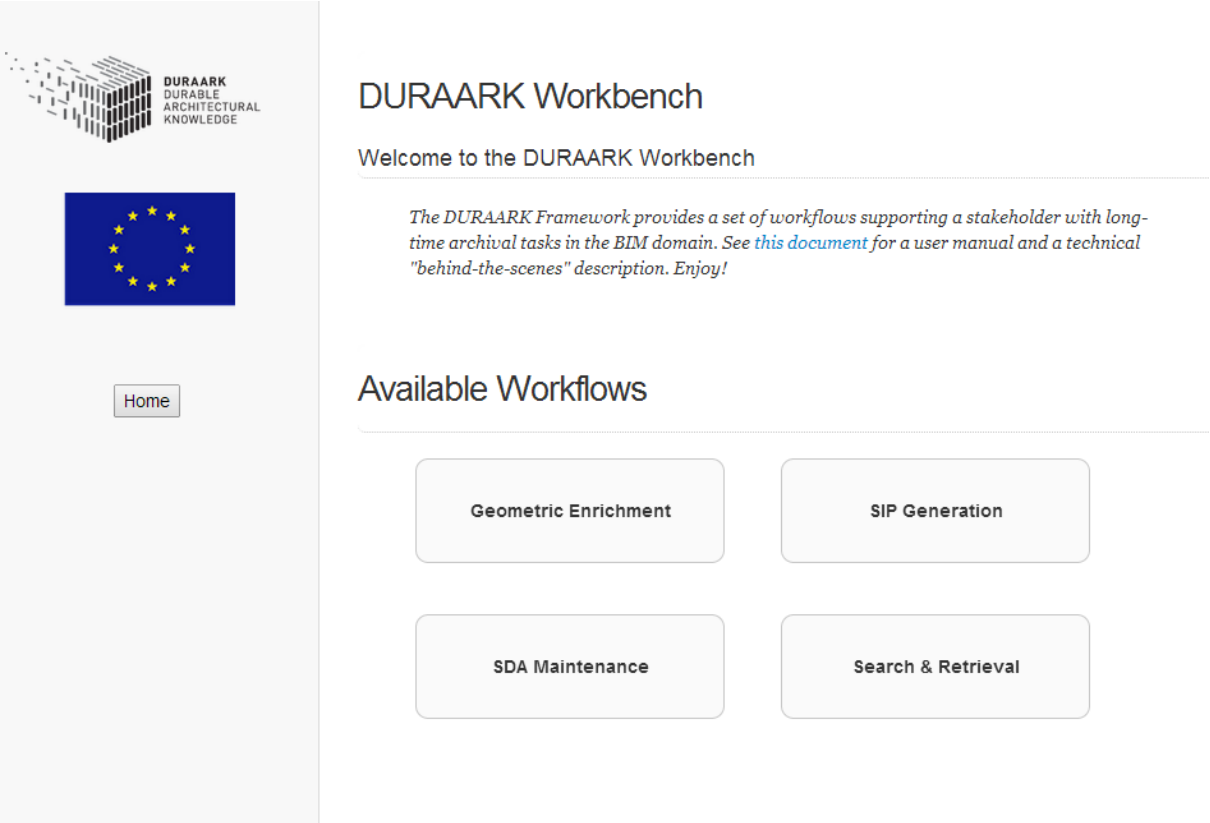

Figure 2: **A screenshot of the integrated *Workbench* software prototype for selecting a workflow.**

The main part of this document is dedicated to the description of these workflows in section 2.1. The remainder of this report is structured in the following way: Section 2 describes the DURAARK Workbench, including the workflow description in form of a user manual. Section 3 sheds light on the architectural design of the workbench. In Section 4 a rationale for design decisions is given, together with a discussion on their risks. Finally a conclusion and impact description is given in Section 6.

Source Code

The source code of the Workbench itself as well as of most individual components is available under an Open Source license and can be accessed at the following URLs:

Workbench `https://github.com/DURAARK/workbench`

E57 metadata extractor `https://github.com/DURAARK/e57Extract`

2 DURAARK Workbench

The **DURAARK Workbench** is a service-oriented platform comprising the software deliverables produced over the life-time of the DURAARK project. The functionality of the deliverables is accessible via a coherent graphical user interface (GUI). The GUI is referred to as the **WorkbenchUI** in the remainder of this section, the service-oriented platform as **Workbench**, where the functionality of the software deliverables are called **Components**.

The WorkbenchUI is the graphical part of this software deliverable allowing a stakeholder to go through a workflow. Section 2.1 explains the intended usage of the WorkbenchUI in form of a user manual. Each workflow is described, accompanied by screenshots of the application. The actual software is available via the URL `http://workbench.duraark.eu` for testing.

The WorkbenchUI is interacting with the components through a service-oriented application programming interface (API) layer. In Section 3.2 a functional description of each component is given, together with the current state of its implementation. Appendix 1 describes and the API to the components.

2.1 User Manual

This user manual guides a stakeholder through the usage of the WorkbenchUI with the description of four workflows:

1. SIP Generation
2. Search & Retrieval
3. Geometric Enrichment
4. Semantic Archive Maintenance

The WorkbenchUI is a web application accessible with a web browser via the URL `http://workbench.duraark.eu`. Workflow 1,2 and 4 are solely running within a web browser. Workflow 3 (Geometric Enrichment) uses the *point cloud registration prototype* from software deliverable **D4.1**, which is implemented as a desktop GUI application. This application has to be installed on the stakeholders computer, the process is described in

the corresponding section of the user manual.

The general user-interaction paradigm for the WorkbenchUI is to first select the desired workflow from the start screen of the application, depicted in Figure 2. The stakeholder is then guided through the individual steps of the workflow. Each step is carried out in a so called **page** (following the web application terminology). Figure 3 shows the general structure of a page: On top there is a "Next/Previous" button bar that moves the stakeholder from one page to the next or previous one. Below the page title and a description of the current workflow step is given. The bottom most section contains the interactive part of the page and/or displays data. The usage of those parts is the focus of this user manual.

Workflow 3 and 4 are special as they provide a selection of tools before starting the user interaction. Depending on the task it is possible that only a single page contributes to a workflow.

The remainder of this section goes through the four workflows and describes each contributing page. When applicable, the component connected to the GUI page is mentioned so that interested readers have the possibility to get a more technical description of the component in Section 3.2 or dive into the description of the corresponding application programming interface (API) in Appendix 1.

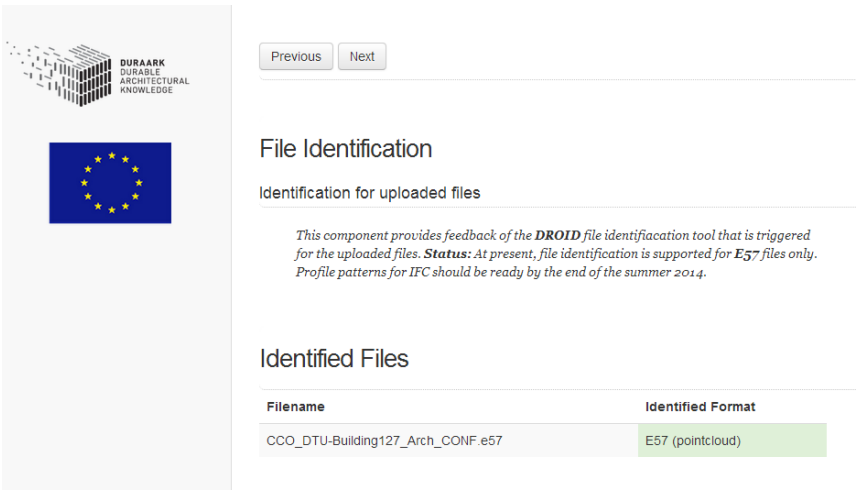

Figure 3: **Page layout example: (top) Workflow navigation (center) Description of the workflow step (bottom) Area for user-interaction and/or data display**

2.1.1 Workflow: SIP Generation

The SIP generation workflow allows the stakeholder to upload data files describing a building which are then packaged into a single SIP file that is ready to be uploaded into a digital preservation system. In the process the files are identified and metadata is added.

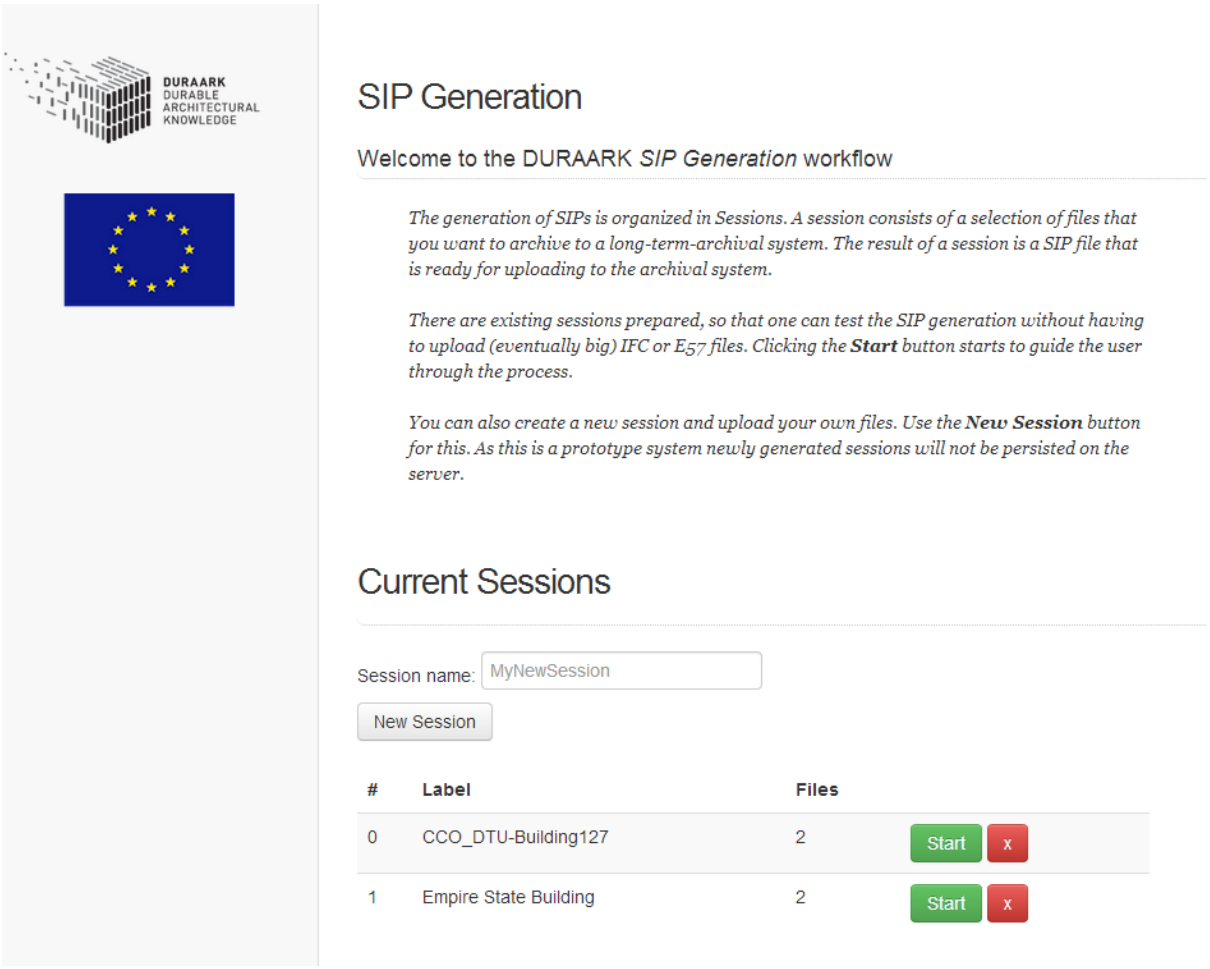

Figure 4: **Session Page**

2.1.1.1 Session Page The SIP generation is organized in so called **sessions**. The stakeholder creates a new session via the *New Session* button after entering a name for it. The session is added to the list on the bottom of the page and can be started via the *Start* button, as well as deleted via the red cross button. The purpose of a session is to a) start a session, work on it and resume it at a later point and b) to allow collaborative working on a session. E.g., Stakeholder A starts a session and provides input files, Stakeholder B

works on the corresponding metadata record.

To allow easy testing of the application one predefined session is provided, already containing input files. When selecting the predefined session the page described next in this user manual (the file upload) is skipped. Creating a new session and starting it opens the **File Upload Page**.

Related content: Appendix 1.1 shows the API.

2.1.1.2 File Upload Page This page allows the stakeholder to upload files relating to the same building(s) into the session. An IFC file and/or an E57-file have to be uploaded. If both file types are uploaded an optional registration file between the two can be selected. The workflow for creating a registration file is described in 2.1.4. For uploading the stakeholder selects the desired files from the computer and presses *Upload*. When the upload is finished - which is indicated via a message - the *Next* button is enabled to continue to the **File Identification Page**.

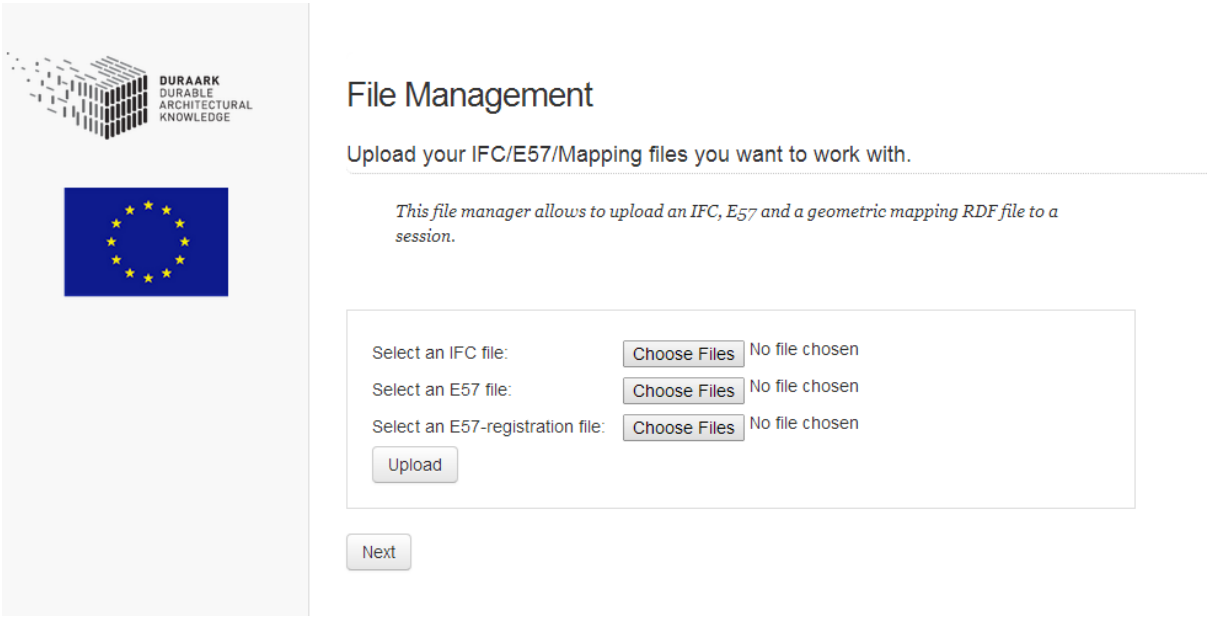

Figure 5: **File Upload Page**

2.1.1.3 File Identification Page If a file of type **E57** is present in the session an identification of the file takes place via the DROID file profiling tool from "National Archives"[1]. Depending on the size of the file this process can take up to a few minutes. The result of the identification is presented in the *Identified Files* section of the page. A green label in the table cell *Identified Format* indicates a successfull identification, a red label an unsuccessful one. In this case the stakeholder is asked to check the uploaded E57 file and upload the corrected file. The purpose of the screen is to prevent the upload of an invalid (E57) file into a long-term preservation system without knowing. Also the following metadata extraction requires a correctly identified file type as input, to prevent follow-up errors in the application. After a successfull identification the stakeholder clicks on the *Next* button to proceed to the **Metadata Extraction Page**.

At present, file identification is supported for E57 files only. Profile patterns for IFC should be ready by the end of the summer 2014.

Related content: Section 3.2.1 gives a deeper look into the used File Identification component. Appendix 1.2 shows the API.

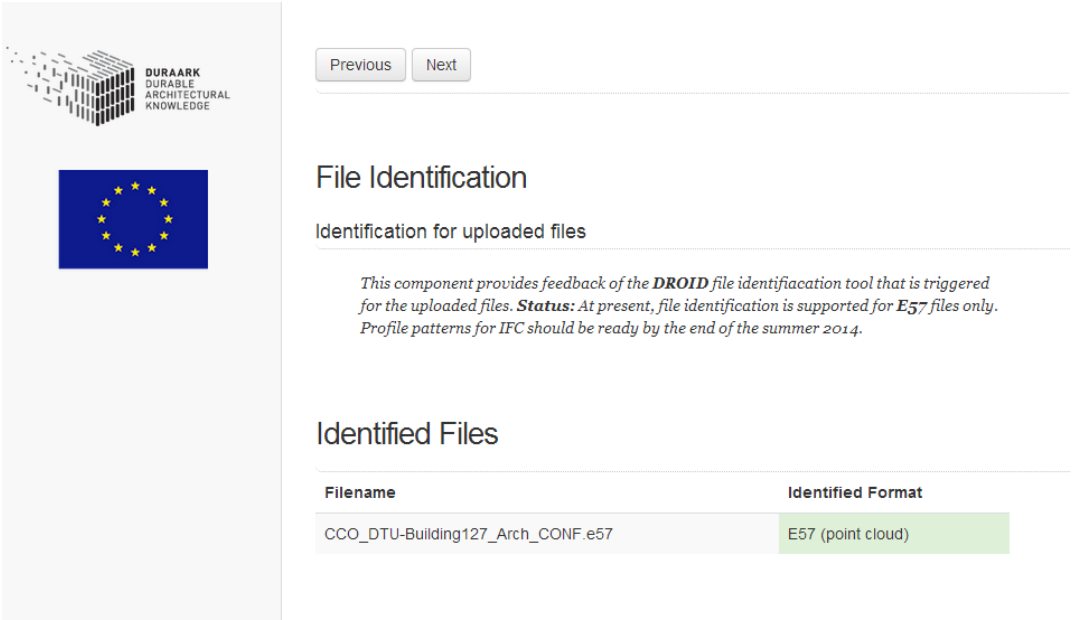

Figure 6: **File Identification Page**

[1]DROID profiling tool: http://www.nationalarchives.gov.uk/information-management/manage-information/policy-process/digital-continuity/file-profiling-tool-droid/

2.1.1.4 Metadata Extraction Page On this page the session is searched for an IFC and a E57 files. If one of them or both are found the metadata for the files is extracted in a background process. The extracted metadata is listed and can be changed by the stakeholder by clicking on the respective cell. If one or more mandatory metadata entries are not present in the IFC file the application automatically adds those entries and colors the entries in red, so that the stakeholder gets a visual hint on which mandatory entries are still missing[2]. Be aware that no validation of entered metadata is taking place at the moment. After changes are made the (appearing) *Save* button has to be clicked to persist the changes. The resulting metadata entries are stored and will be added to the final SIP file in form of an RDF Turtle file.

Related content: Section 3.2.2 gives a deeper look into the used E57 metadata extractor component. The IFC metadata extractor is described in **D3.3**. Appendix 1.4 and Appendix 1.3 show the API.

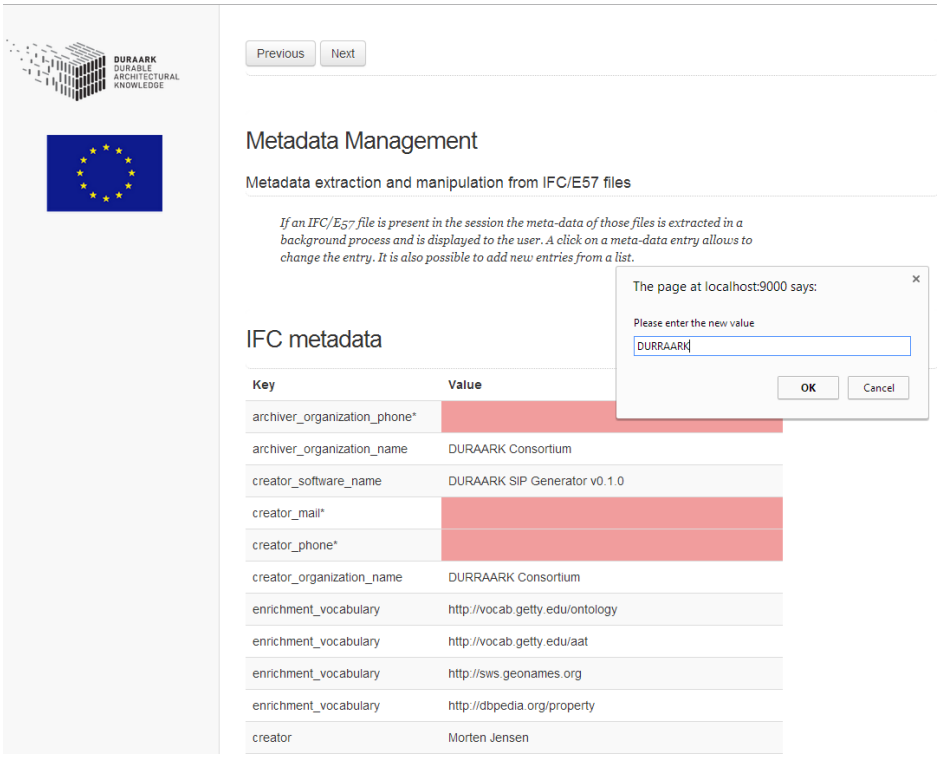

Figure 7: **Metadata Extraction Page**

[2]In this version of the application the definition of mandatory metadata is not finalized yet and will change in future versions.

2.1.1.5 Semantic Enrichment Page This component uses metadata extracted from the ingested IFC file to search for addtional information which the session will be enriched with. The search is able to incorporate different sources from the available metadata (e.g. city names). This version of the Workbench is taking the postal address in the metadata as search criteria. The page shows a list of the related linked open data (LOD) sets and is stored within the RDF file that goes into the SIP file at the end of the workflow.

Future versions will allow a more fine-grained control over the enrichment process, as well as manual modification of the found data-sets.

Related content: **D3.3** gives a deeper look into the used Semantic Enrichment component. Appendix 1.5 shows the API.

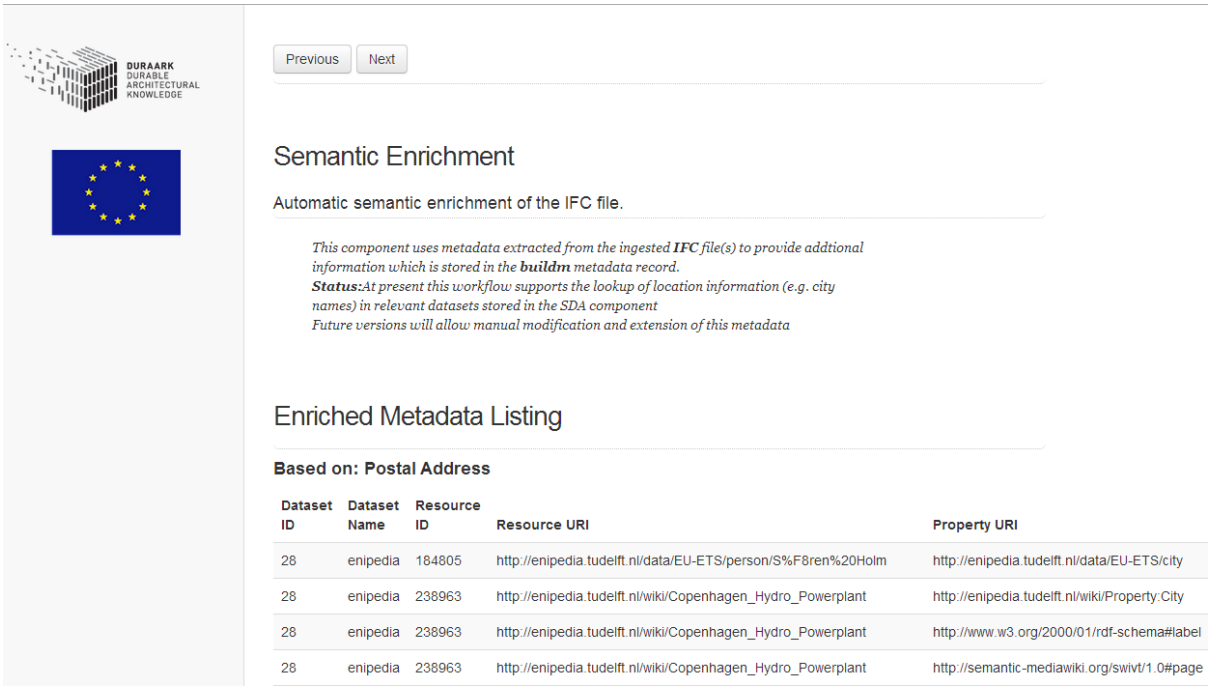

Figure 8: **Semantic Enrichment Page**

2.1.1.6 SIP Generation Page This page presents all files that will be packaged in accordance to the implementation specifics of the digital preservation system (DPS). The engineering metadata from the extraction of the IFC and E57 files together with the descriptive metadata from the enrichment process are put together into a single RDF file ("buildm.ttl") that goes into the archive. This includes the structuring of all digital objects

and the metadata records into a METS file in accordance to the specification of the vendor.

The *Content Overview* lists the package archive with file names, sizes and type. Clicking the *Generate SIP* button starts the background process to generate the archive. In this process a mapping from the metadata RDF file to the METS structure is done yielding in a *sip.xml* METS file. The resulting archive is a *ZIP* file with the *sip.xml* and a *content* folder as root items. The content folder contains the uploaded files together with the RDF metadata file. The generated ZIP file is of version 2.0, for which we recommend the free software **7zip**[3] for opening the archive. Tests showed that the integrated ZIP archive handler in Microsoft Windows 7 was not always capable of opening the valid archive.

Depending on the file size of the session files this process takes up to a few minutes. After a successfull creation of the archive the SIP can be downloaded via the appearing *Download SIP* button. Hidden from the user the generated SIP file is passed over to the PROBADO3D-Rosetta Connector[4] (see Section 3.2.4) which creates an entry in the PROBADO3D component's internal database (see Section 3.2.5 to allow the stakeholder to search for the metadata of generated SIPs later on (see 2.1.2 for the workflow description).

This page finishes the **SIP Generation** workflow and yields a SIP file that is ready for uploading to the digital preservation system. The actual upload is target in future versions of the Workbench. The SIP package will be targeting the commercial **Rosetta** DPS then.

Related content: Section 3.2.3 gives a deeper look into the used SIP Generation component. Appendix 1.6 shows the API. Section 3.2.4 explains the PROBADO3D-Rosetta Connector component, Section 3.2.5 the PROBADO3D component.

[3] 7zip download URL: `http://www.7-zip.org/download.html`

[4] Despite the name the component is not yet deriving it's input data from the Rosetta system, but works directly on the SIP file. This will change in future versions of the Workbench.

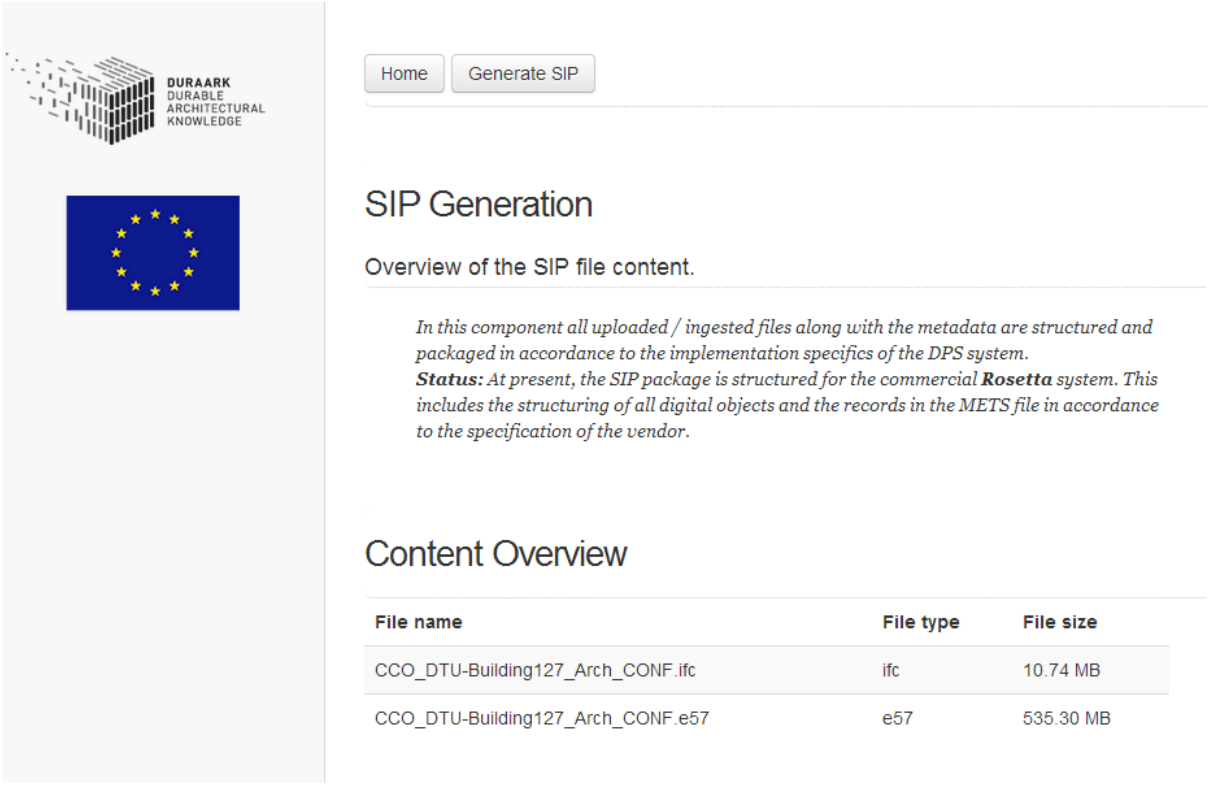

Figure 9: **SIP Generation Page**

2.1.2 Workflow: Search & Retrieve

This workflow provides the stakeholder with the possibility to search for metadata that was ingested into the PROBADO3D database via the SIP generation workflow. PROBADO3D is a content-based indexing and retrieval service for non-textual documents, e.g. for BIM related (meta)data. Keep in mind that the generated SIP is not persisted at the moment, as this is the task of the DPS which will be integrated in future versions; only the metadata is). The page starts with the listing of all generated SIP creation events and allows the stakeholder to inspect the corresponding metadata. The *Search* field provides a filtering method. The stakeholder enters a search term resulting in a full-text query over all metadata entries. The resulting SIP creation events are listed.

Related content: Section 3.2.5 gives a deeper look into the used PROBADO3D component. Appendix 1.7 and Appendix 1.8 show the API.

2.1.3 Workflow: Semantic Archive Maintenance

This workflow includes two tools, which are selectable via the **Semantic Archive Maintenance** start page:

- SDO Information
- Dataset Crawler Module (which is described in **D3.3**. Please refer to that document for a user manual.)

The **SDO Information** tools allows to lookup information that is stored in the *Semantic Digital Observatory (SDO)*. The SDO component discovers and retrieves suitable architecture-relevant datasets in crawling linked open data sources and provides structured metadata on those datasets. The *Dataset Crawler Module* is part of the SDO and performs the actual crawling of data. A detailed explanation to both can be found in **D3.3**, here the GUI integrated in the WorkbenchUI is described.

Figure 10 shows the SDO Information page. The stakeholder is provided with a list of data sources which are used for crawling linked open data. A name, description, URL and last crawl date is displayed for all the endpoints. The *Search Topic* box allows searching for specific data-sets in all of the listed end-points and after clicking the *Search* button a list with the results is displayed.

Related content: Report **D3.3** explains the SDO and the Dataset Crawler Module.

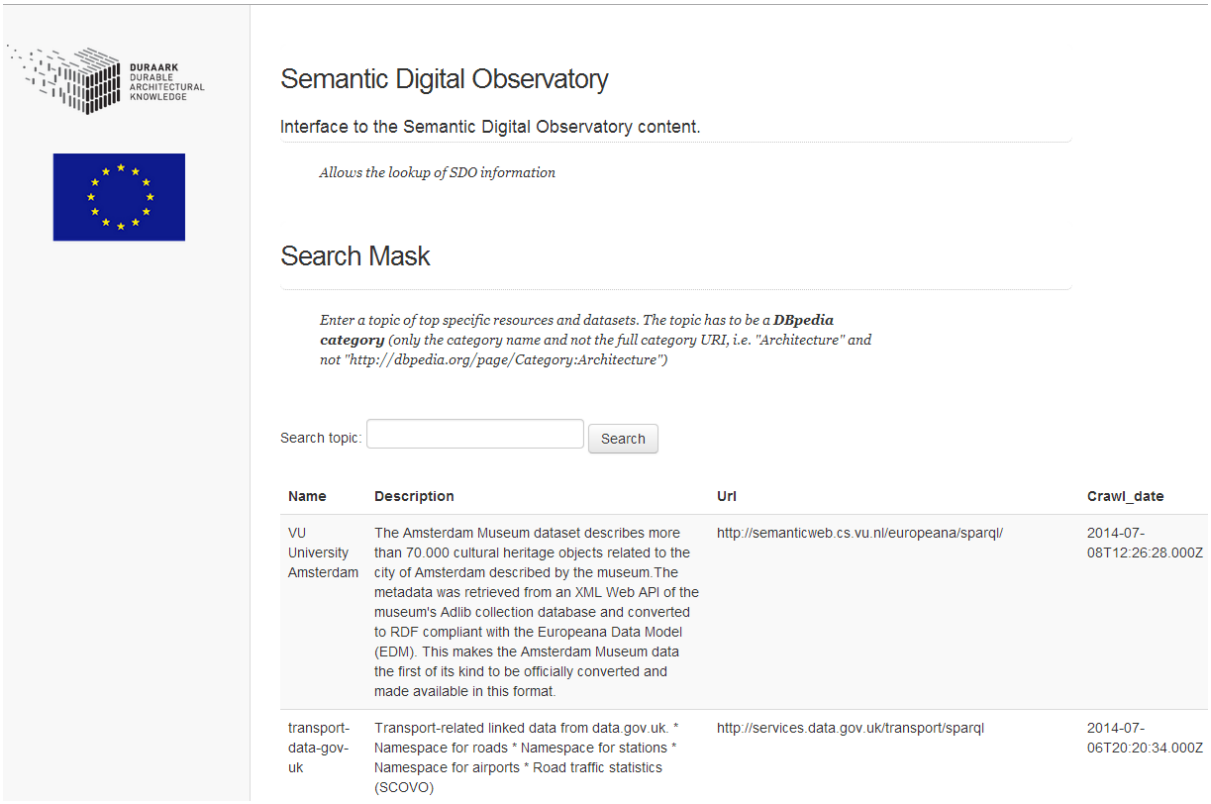

Figure 10: **List of SDO endpoints and search mask**

2.1.4 Workflow: Geometric Enrichment

The start page for the geometric enrichment (see Figure 11) shows a list of available point cloud tools. In this version the "registration prototype" from the D4.1 software deliverable is available and can be selected. This software is a standalone desktop application, which needs to be installed before the first usage. If the software was not installed yet the stakeholder is provided with a download link and installation instructions. After a successfull installation a click on the icon opens the **Session Page** (see 2.1.1.1). Here the stakeholder can select one of the existing sessions or creates a new session. A click on the *Start* button of an existing session starts the download of the IFC/E57 files denoted in the session. After a successfull download the registration prototype opens with the downloaded files as input. Figure 12 shows the appearing GUI with the two selected files loaded. At this point, the reader is referred to **D4.1**, Appendix A, for a description of the usage of the registration software prototype.

After the registration process is finished the resulting mapping RDF file has to be stored on the local harddrive. This file serves as input file for the SIP Generation workflow described in Section 2.1.1.

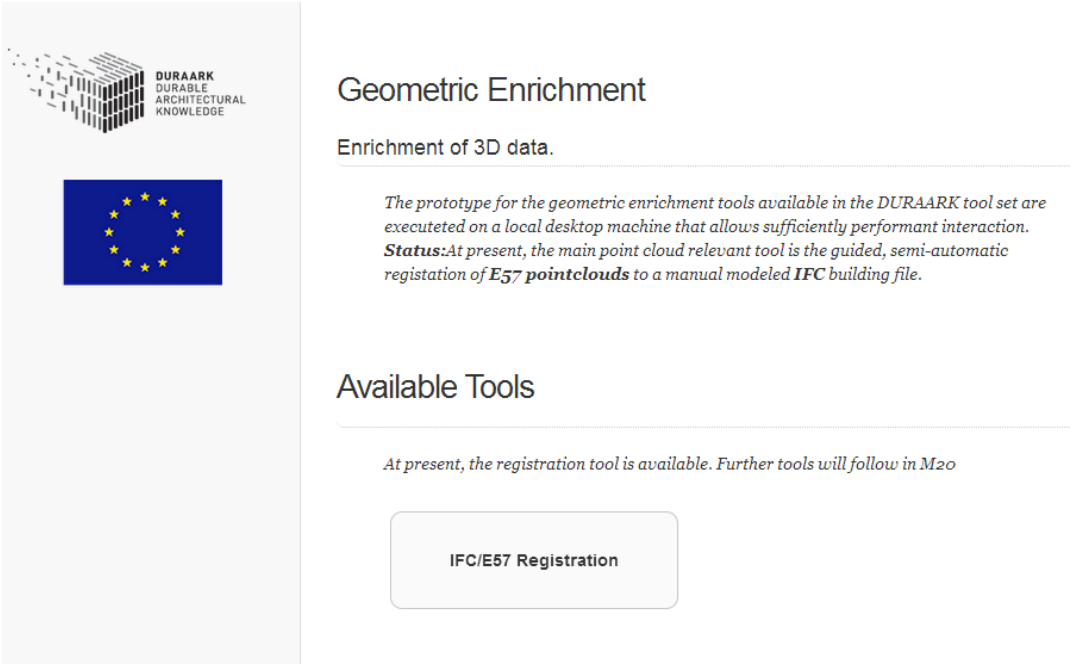

Figure 11: **Geometric Enrichment tool selection page**

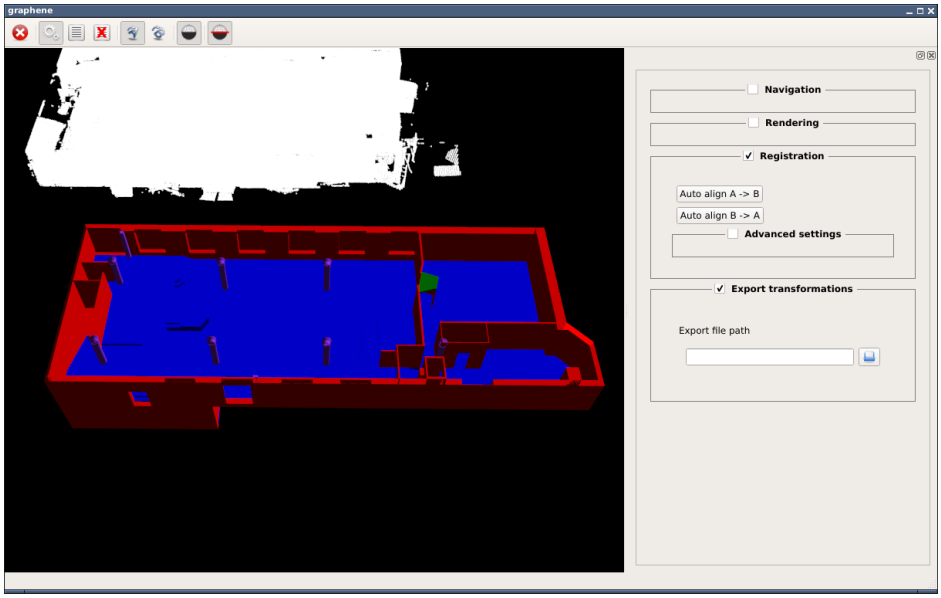

Figure 12: **Start page of the registration prototype**

3 Technical Implementation

3.1 Software Design

When developing web applications, in many cases their structure is following a common pattern that consists of three layers: a *frontend* layer containing the user interface logic and the display of data (the GUI), ii) a *backend* layer that processes and provides data and iii) a communication layer between those two. The frontend layer is located in the user's web browser, the backend layer is running on a server host accessible via the internet. The connection layer is a data exchange protocol that transports data over a network connection, e.g. a RESTful API[5].

For the DURAARK project it is necessary to integrate different components from partners into a coherent, integrated software prototype. The input and output characteristic of the developed components is suited to be mapped to the described common pattern. For instance, to upload data to the DPS it is first necessary to select the files that should be persisted. The user selects files in the web browser, which is happening in the frontend layer. Those files are then uploaded to a web server and are checked for the correct file type, which is happening in the backend layer. The other components developed in DURAARK (see 3.2 for a list and the respective descriptions) fit this pattern, too.

As a consequence and for having a platform for connecting the heterogeneous components the decision was taken to develop a general framework - the DURAARK Framework - providing a sound base for developers in the project and possible future (third party) developers. The vision for the DURAARK Framework is to provide a future-proof, extensible and light-weight software library for building web applications focusing on long-term archival of data.

[5] See http://www.infoq.com/articles/rest-introduction for an introduction to REST and RESTful APIs.

3.1.1 Overall Architecture

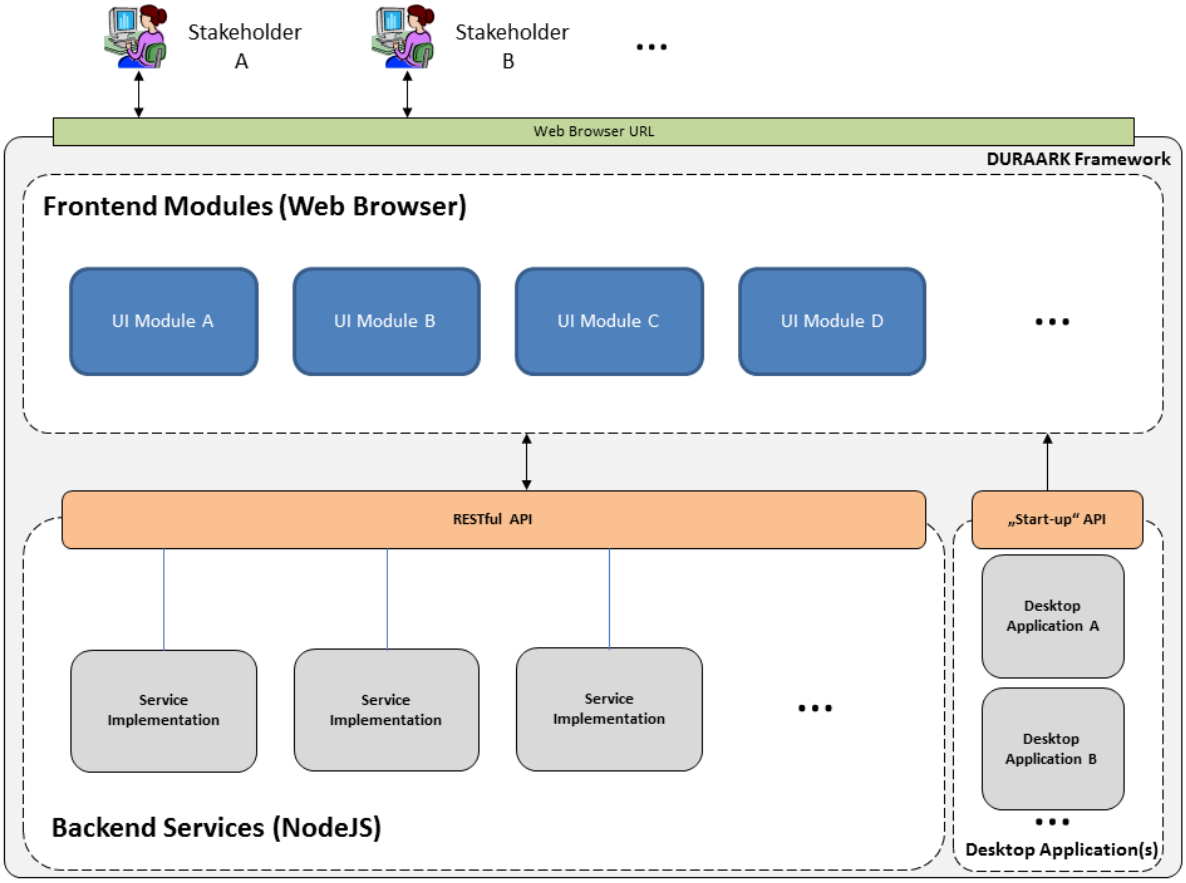

Figure 13: **Integration design diagram for the DURAARK framework**

The overall architecture is directly derived from the web application pattern described in Section 3.1. On the frontend side, so-called *User Interface (UI) Modules* are responsible for displaying data and interacting with the user. On the backend side *Web Services* are processing data and deliver the data in a consumable form for the UI modules. The web service layer of the DURAARK framework provides a RESTful API to communicate between service and UI module. The actual implementation of the web service has to be provided by the developer. This decoupled approach makes it easy to exchange the implementation of a web service with another or updated one, without having to change a) the code in the consuming UI module and b) the API code of the web service.
Figure 13 shows the overall architecture. The framework holds a list of UI modules which can be registered to the system, allowing a central module management with version

control, automatic update mechanisms, access control, etc. On the backend another module management component keeps track of all the registered web services. The UI modules communicate with a web service via a RESTful API. The API handles a request from a UI module and delegates it to the web service implementation, which in turn delivers the requested data back.
The architecture allows for two communication scenarios between UI modules and web services. First, a UI module is directly communicating with a web service, which is the case when the web service needs to be configured by the user (e.g. in entering metadata that the service processes then). The second scenario covers the direct communication between two web services. This is the case if - for instance - the service responsible for the generation of a SIP package asks the service for file identification to verify, if a file has the correct type before creating the package. In both cases the defined RESTful API is the enabler for this kind of application-to-application communication.
A stakeholder is interacting with the frontend part of the framework. She does not have to know anything about the web services that are doing the actual work (data) processing. Also, the web services (as the name suggests) can be distributed over the network, it is not necessary for them to reside within a single server context. For instance, the services related to the SDA (see **D3.3**) and the PROBADO3D service are running on different servers than the rest of the components, which are located on a single server in the current setup.

3.1.2 Frontend - User Interface (UI) Modules

A User Interface (UI) Module is a visual page within the web browser that is a) displaying data and b) allows for interaction with the user. The technology stack consists of HTML and CSS for the visual representation of data and Javascript for the user interaction logic. The DURAARK framework is using an existing Javascript library that is tailored for presenting data in a web browser and for manipulating this data. The library is called *Backbone.Marionette*[6]. Backbone.Marionette provides the basic tools for a structured development of user interface logic, is actively developed and has a broad and active community.

[6]MarionetteJS: http://marionettejs.com

3.1.3 Backend - Web Services

This part is the *backend* of the framework running on a server. It provides the developer with base classes that cover common functionality used when developing web services for the DURAARK project. The base classes allow to create a RESTful API around *standalone executables* that have a file as input and produce a) an output file or b)console output for further processing. For instance, one component takes care of file identification. The component is available as a standalone executable and needs an IFC or E57 file as input. Its output is a description file that contains informations for the provided file. This is a typical processing step for web services in the DURAARK context, which is common for other components in the project, too.
The framework supports the developer in creating a RESTful API around a given functionality (e.g. a standalone executable). The implementation providing concrete functionality (e.g. the file identification component) is exchangeable, whereas the API does not have to be changed when used with another implementation of the service. This approach encourages a stable API development and a clear separation of concerns between service interfaces and their implementation.
As a basis for the web services part of the DURAARK framework the software library *NodeJS*[7] is used, which provides the functionality to start a web server and handles requests from and responses to clients (e.g. a UI module). It is written in Javascript and is a stable and well-tested software library with a broad user community and an active development line.

[7]NodeJS: http://nodejs.org

3.2 Components

Components are the functional parts developed in the project which are accessible as web services. The DURAARK framework provides the infrastructure to connect the graphical user interface of the integrated prototype (the "WorkbenchUI") with the web services via an RESTful API. The tools for the geometric enrichment workflow (currently containing the D4.1 software deliverable application) are the second type of components. Those are graphical standalone applications which are not reasonably transferable to a web service implementation at the moment, as they require graphical user interaction that is not easily done via a UI module because of the web browser runtime environment. For this reason the DURAARK framework provides the possibility to "start-up" the tools via a UI module (see 2.1.4). The stakeholder uses the application and produces a result, which in turn is again handled by a corresponding workfow in the WorkbenchUI. In the current version the D4.1 "Documenting the Changing State of Built Architecture" application (in short: registration prototype) produces a geometric mapping file between IFC/E57 files. The IFC/E57 input files are determined by the stakeholder via the WorkbenchUI (meaning that DURAARK's web services, as well as the developed desktop applications are working on the very same files) and the produced output file is used in the SIP Generation workflow as input file (see Section 2.1.1).

The approach of separating a service from its user interface is a powerful mechanism to enable new and existing applications (eventually written in other programming languages than Javascript) integrate the DURAARK functionality, as a RESTful API is client-agnostic. This way the GUI is completely independent from the service implementation. Figure 14 shows the DURAARK framework approach on how to connect the UI modules of the WorkbenchUI with the web service and desktop application components.
This section gives an overview on the *components* that are developed within the DURAARK project and which are not already described in report **D3.3**. D3.3 includes the Semantic Digital Archive (SDA), the Semantic Digital Observatory (SDO) including the Dataset Crawler Module mentioned in the user manual section 2.1 and the component for the semantic enrichment of an IFC file. This report **D2.4** gives an overview of the technical implementation of the following components:

- File Identification
- E57 Metadata Extraction

- SIP Generator
- Rosetta-PROBADO3D Connector
- PROBADO3D
- Geometric Enrichment tools

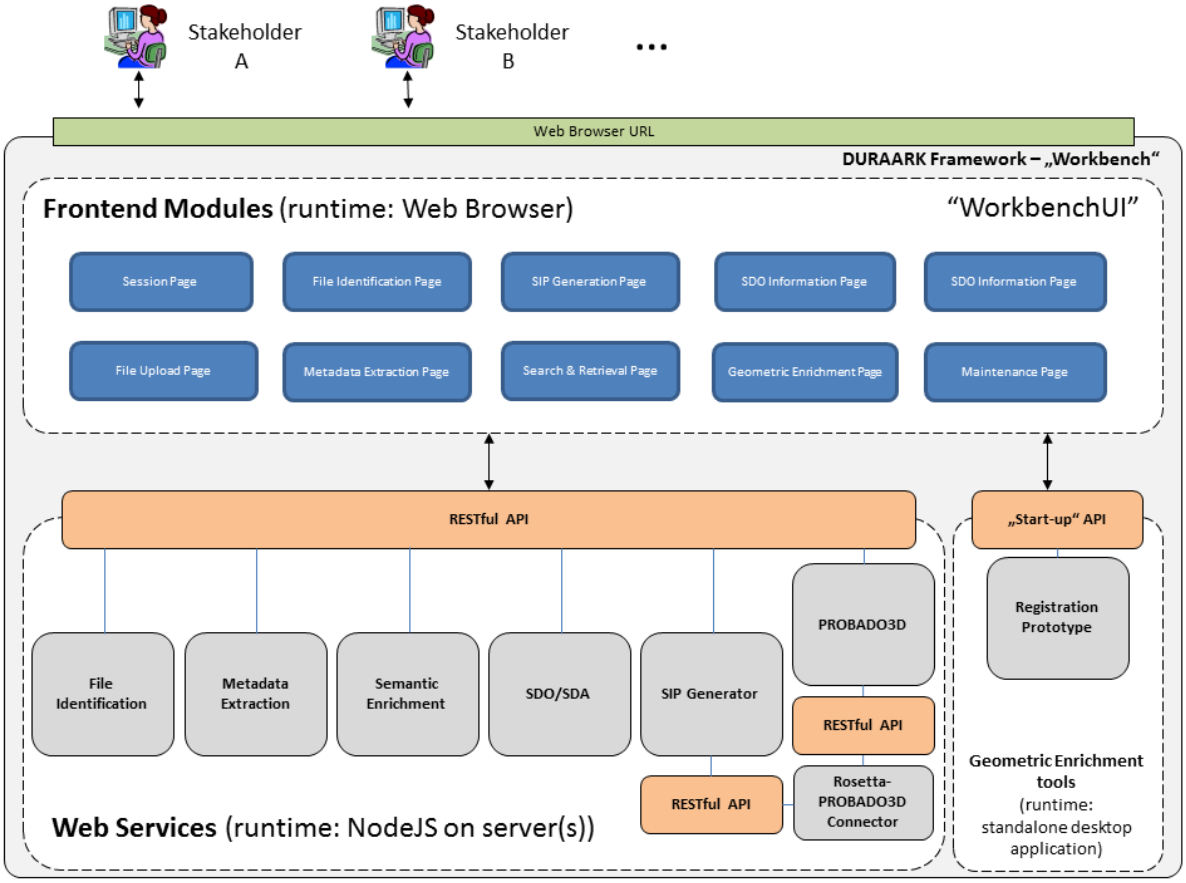

Figure 14: **Architectural diagram for the Workbench platform**

3.2.1 File Identification

Since the exact file format identification is needed for preservation planning of the ingested files, the widely used file format identification tool from the National Archives **DROID (Digital Record and Object IDentification)** was chosen for the DURAARK

Workbench[8]. DROID is developed for archives and institutions which have to identify file formats for their stored objects. It identifies formats based on patterns (e.g file format extension, internal IDs, etc.) and is updated constantly through xml-based signature files which provide the linking to the entries in the PRONOM technical registry with its assigned PUID (Pronom Unique IDentifier).
Figure 16 show the integration of the component into the Workbench.

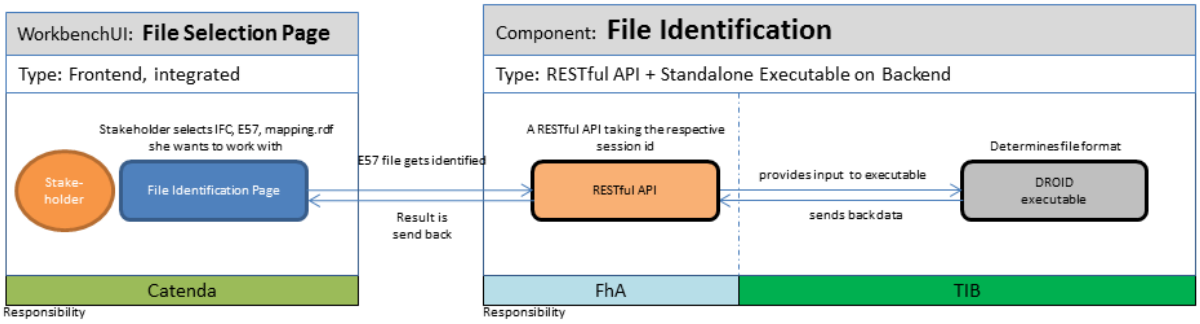

Figure 15: **Integration design diagram for the File Identification component**

3.2.2 E57 Metadata Extraction

The E57 metadata extractor is a shared library written in C++ which uses **libE57** at its core to parse E57 point cloud files and extract meta-information like - for instance - the number of scans, number of points, acquisition date, dimensions of embedded images, etc. In addition to the library, a command-line tool is provided which exposes the library's functionality. This command-line tool may be used as a stand-alone component for metadata extraction without having to link the library code directly into another component by calling the executable from another process. When the tool is executed with the "--help" argument, a concise usage guide is printed. Otherwise, the tool must be given at least an input E57 file using the "--input" parameter. The output may be either written to a file using the "--output" parameter to specify the output file path, or – if no "--output" parameter is given – written to standard output for piping it to another process. The desired output format may be specified using the "--format" parameter which can have either “json” or “xml” as its value; JSON is the default if no format is specified. The extracted metadata is output in a structured, hierarchical format so that it may be further processed by other components, or for ingest into the archive alongside the E57 data files.

[8]DROID profiling tool: http://www.nationalarchives.gov.uk/information-management/manage-information/policy-process/digital-continuity/file-profiling-tool-droid/

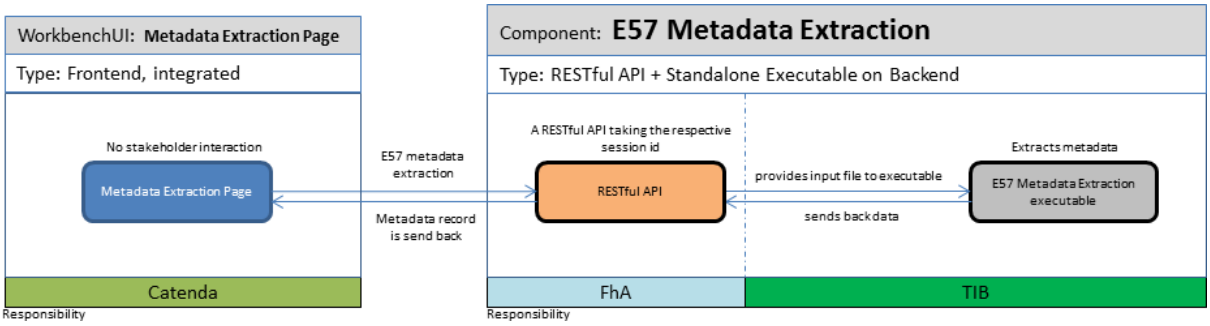

Figure 16: **Integration design diagram for the Metadata Extraction component.**

3.2.3 SIP Generator

A digital archive must have features and methods to receive and manage digital content. This should, wherever possible, be done in an automatic process which means that digital objects should be delivered in a structured and standardized way. In order to achieve this, a software is developed within the DURAARK project that generates a Submission Information Package (SIP) to be delivered to a DPS. The SIP generator software will support producers with the process of compiling digital assets to be ingested to a digital archive.

Input to this module consists of both manually entered data by the producer/user captured by the GUI, uploaded files and automatically captured metadata such as file identification results consisting of e.g. unique id, size and hash sum.
The SIP generator is written in Java and is using a database for temporary storage of meta data.
Figure 17 shows the sequential actions taken to generate a SIP package, Figure 19 shows the integration into the Workbench.

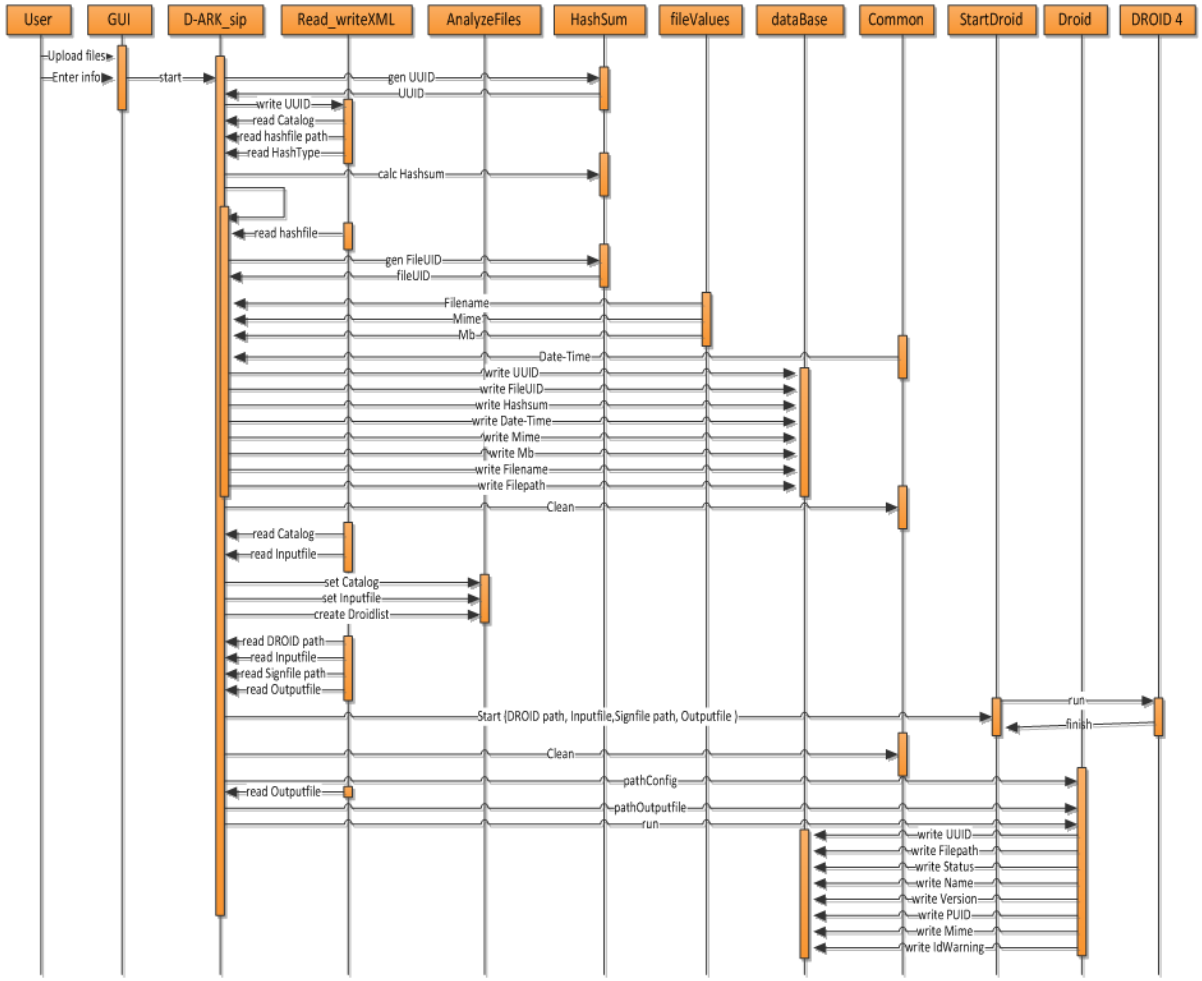

Figure 17: **Sequence diagram for the SIP generation**

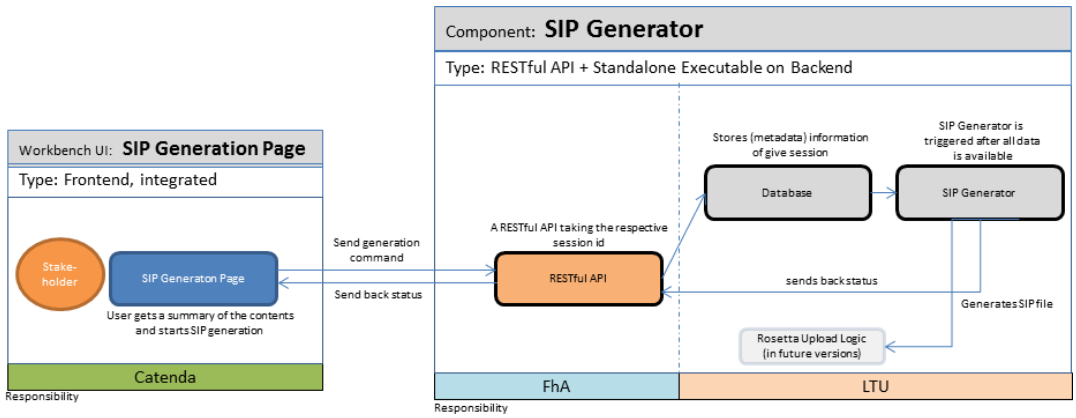

Figure 18: **Integration design diagram for the SIP Generator component**

3.2.4 Rosetta-PROBADO3D Connector

In general the Rosetta-PROBADO3D Connector is responsible for indexing data uploaded to the Rosetta system. Rosetta is providing a REST interface that allows to access the uploaded data. The connector will be utilizing this interface to request the metadata that is necessary for the indexing process, so that the user can search the dataset later on. However, for this M18 software prototype the Rosetta system is not targeted yet. Therefore the connector is taking the RDF metadata file generated via the SIP generation workflow indexes the data directly from the that file, instead of requiring the same information from the Rosetta REST interface.
Internally a new dataset entry is created for each generated SIP. The dataset is filled with the given metadata stored into the internal database of the PROBADO3D system.

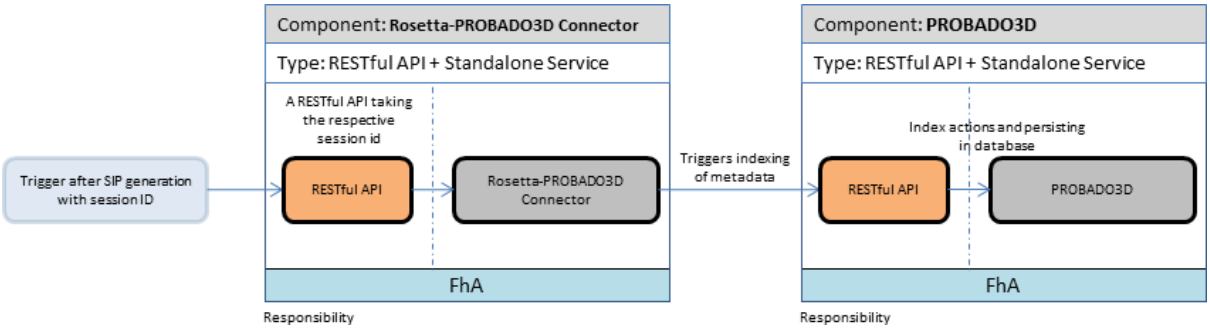

Figure 19: **Integration design diagram for the Rosetta-PROBADO3D Connector component**

3.2.5 PROBADO3D

The PROBADO framework allows integration of content-based indexing and retrieval methods for non-textual documents. The PROBADO3D architecture follows a three layer approach which consists of a repository layer, a core system layer, and a presentation layer. Distributed local repositories implement document-type specific indexing and accessing techniques, including rich meta data models. The PROBADO3D core layer keeps track of all document repositories registered in the system. It maintains an integrated index of all documents. The presentation layer offers rich user access methods, including graphical query specification, and document visualization. PROBADO defines a system protocol based on web service technology. It allows dispatching content-based and metadata-based user queries to local repositories, which manage the primary documents. Synchronization

methods allow the repositories to inform the core system about availability and updates of hosted contents.

PROBADO3D is used as the interface for browsing and searching the generated SIP items. This can either be done by using the PROBADO3D web pages or the various web interfaces provided by the PROBADO3D service. PROBADO3D is especially tailored to the needs of the architectural domain and establishes a search & retrieval infrastructure (e.g. indexing, 3D PDF preview generation, etc), which can be easily utilized for the various DURAARK needs.

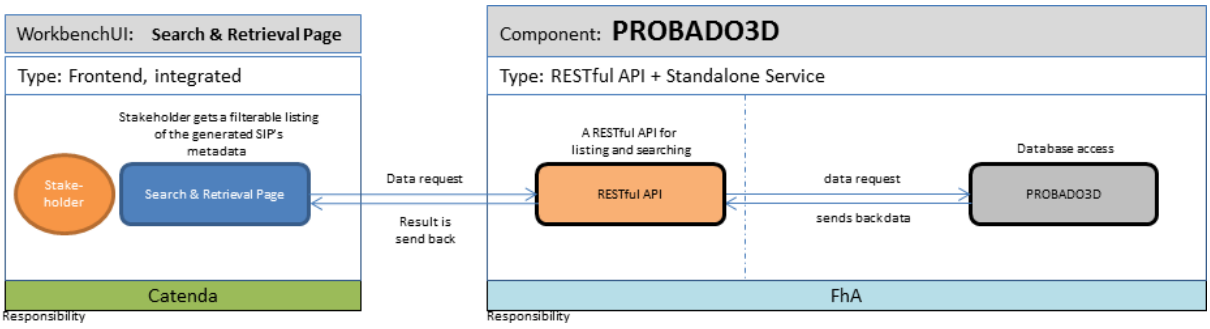

Figure 20: **Integration design diagram for the PROBADO3D Search & Retrieval component**

3.2.6 Geometric Enrichment

The components developed in WP4 and WP5 are implemented as standalone desktop tools which do not directly connect to remote services but instead process files residing on the user's (client) computer. Their main purpose is the enrichment of datasets before the actual ingest takes place. For the first DURAARK system prototype, we have focused on the integration of the registration prototype (**D4.1**) for demonstrating the workflow using standalone desktop tools; other software prototypes of WP4 and WP5 will work in a similar manner. Figure 21 shows an overview of the registration component's input/output specification.

The envisioned workflow for using the registration component is as follows. During the preparation of the ingest of multiple new datasets of the same building using the WorkbenchUI (for instance multiple scans taken at different points in time or a point cloud and a corresponding BIM model), the user has the opportunity to select a pair of datasets which shall be registered (i.e. spatially aligned) to each other. It is assumed

that the datasets are available as local files during the ingest process. After selection of the datasets, the registration prototype (i.e. the software prototype executable) may be started from within the WorkbenchUI which is automatically provided with the file paths of the selected files as command line arguments. These paths are used to initialize the file chooser which is presented to the user by the executable.

At this point, the reader is referred to D4.1, Appendix A, for a description of the usage of the registration software prototype.

After the datasets have been registered and the resulting mapping has been exported as an RDF file, the exported file may be selected/uploaded to the WorkbenchUI for inclusion in the SIP generation workflow (see 2.1.1).

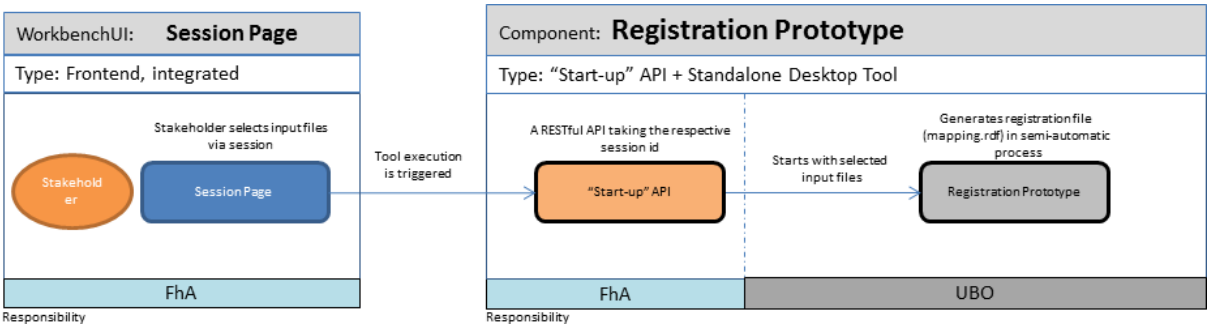

Figure 21: **Integration design diagram for the registration component.**

4 Decisions & Risks

4.1 Technical decisions and impacts

Web-based user interface for the integrated prototype

The software's graphical user interface ("WorkbenchUI") is developed with a web technology stack running in a web browser. The browser environment implies advantages over a standalone desktop application, the most important one being the platform independence of the application. A web browser provides a standardized environment[9]. developers can work with. This environment is (to the most degree) the same on different platforms, e.g. Microsoft Windows, Linux and MacOS, but also for the very popular mobile platforms Android, iOS, Windows Mobile, etc. This has the tremendous advantage that when developing an application with a web technology stack it will automatically be usable on the most popular desktop and mobile platforms, without the need to change the application code.

Developing against a browser environment has restriction that are relevant for the DURAARK context. The data sets stakeholders will work with can be huge in size. The web based Workbench is running on a remote server and it is necessary to transmit the files from the local harddisk to the remote server, where the different services have access to them. Even with a reasonably decent network connection an upload of a file that is hundreds of mega-bytes in size takes multipe minutes or even hours.

NodeJS as runtime enviroment for web services

The web services developed in DURAARK are contained within a NodeJS environment. Their purpose is to wrap standalone executables or other web services developed in the project and provide a RESTful API for accessing their functionality to a GUI layer or other services (e.g. application-to-application communication). The wrapper layer is rather thin. It takes care of starting an executable or web service and processing its output so that it is consumable by a client. NodeJS is a reasonable choice for a server backend as it has become very popular in the last years as it is easy to program, provides

[9]Client-side web standards are organized in multiple standard bodies and working groups. The most prominent ones are the World Wide Web Consortium (W3C, http://w3.org/ and the Web Hypertext Application Technology Working Group (WHATWG, http://www.whatwg.org/

a scalable architecture and has a large community that adds a lot of useful functionality in form of modules. The used programming language is Javascript, which is consistent with the UI module programming language. The advantage is that for programmers familiar with Javascript on the browser side the entry hurdle for developing Javascript based web services is low. The knowledge of a single programming language allows to write user interface logic *and* web services for DURAARK.

RESTful API

The service-oriented architecture of the DURAARK framework separates functionality provider from the respective user interface(s). The communication layer was chosen to be a RESTful API[10]. REST means "Representational State Transfer" and is a way to implement heterogeneous application-to-application communication (also including the communication with a user interface module).

With a RESTful API the definition of the REST principles (see a simplified explanation of them in Appendix 2 already gives a developer a lot of knowledge about the provided interface. How to access to the API is recommended (though not standardized) via the use of *HTTP verbs* (e.g. GET for retrieving information, POST for creating new entities, PUT for updating existing entities) that already have a semantic meaning. The second principle of REST is the use of *Unified Resource Identifiers* (URIs) which uniquely identify a provided entity or resource (e.g. a "session" in DURAARK), which can be shared or bookmarked.

The RESTful API allows to access the functionality developed in DURAARK to be accessed by existing or new application which are not implemented in Javascript. The only prerequisite for accessing the API is a network socket, which is available in all relevant programming languages.

JSON as data exchange format

A RESTful API is capable of answering request in different formats, representing the same information, e.g. XML, JSON or a custom format. In DURAARK, JSON is the

[10]See http://www.infoq.com/articles/rest-introduction for an introduction to REST and RESTful APIs.

chosen exchange format. JSON means "Javascript Object Notation" and was developed for the Javascript language to exchange data in a standardized way[11]. As the frontend and backend logic in the project is Javascript, JSON is a natural choice to exchange data between the web services and the UI modules. Every Javascript implementation includes tools for parsing and reading JSON out-of-the-box, making it very easy to use the format. When using other programming languages to access the DURAARK web services tools are available to handle JSON in those languages.

4.2 Risk assessment

This section gives a summary of the Impact section in listing the discussed technical risks, consequence and treatment action:

Risk Description The development of web technology based applications loses momentum, resulting in an unsupported development stack.

Risk Assessment .

Impact High

Probability Low

Description Currently the web browser and the corresponding web technology stack is gaining much attention in application development, mostly because of the advantage of platform independency in the context of mobile development. The probability is rather low that the web technology stack is abandoned in the future.

Contingency Solution WP2 is closely following the developments of web technologies. If the momentum gets lost the endorsed technology will be evaluated and a plan for porting the existing software will be made. Because of the modular design of the DURAARK framework a change to existing and well-established technology stacks (e.g. Qt/C++, XAML/C#, Swing/Java) would be possible, too.

---

[11] JSON explanation and standard description: http://json.org/

Risk Description Javascript as the main programming language for backend and frontend is not accepted by the community.

Risk Assessment .

Impact High

Probability Low

Description In a community it is possible that multiple programming languages are used by respective programmers. A wide-spread myth (though with decreasing tendency) blames Javascript as a non-compatible language compared to Python, Java, etc., which results from the moved Javascript history.

Contingency Solution If the community is not adopting the Javascript-based approach of the DURAARK framework it is still possible to use the existing functionality via the RESTful API. Adding a new web service is possible as providing a RESTful API to a functional block does not demand a Javascript implementation and can be achieved in any other language. The DURAARK project endorses to develop of modular backend functionality and exposing it via a well-defined API. The only disadvantage is that the respective developer can not use the already existing DURAARK framework.

The integration os new UI modules which are not based on a web technology stack is supported, DURAARK is already integrating standalone desktop applications which are not web based.

Risk Description incl. Cause Javascript is too slow for either a user interface or web service task as it is an interpreted, CPU bound language.

Risk Assessment .

Impact Medium

Probability Medium

Description Javascript is an scripting language executed by an interpreter. Compared to compiled languages like C++ or Java an interpreted language is slower per design.

Contingency Solution For the user interface a CPU intense task can be delegated to a backend web service. On the backend a CPU intense task can be handled in the programming language of choice and then wrapped via NodeJS.

Risk Description The stakeholder has no or slow access to the internet, the web application can not be executed, file uploads take too long.

Risk Assessment .

Impact High

Probability Low

Description As web application the DURAARK Workbench heavily depends on a internet connection with reasonable bandwith for a) accessing the application and b) for uploading files to the web services. A non-existing connection prevents the usage of the software, a slow connection reduces the user experience dramatically.

Contingency Solution The M18 version of the prototype is a pure web application and will not work without an internet connection. However, projects exist that allow to convert existing web applications into a standalone desktop application[12][13], where the majority of existing source code can be reused without additional programming work. WP2 will look into this projects to assess their capabilities for producing a desktop application als alternative to the current web application. This would remove the necessity for an internet connection and long upload times for large files, as the services working on the files will run locally on the users computer with access to the local files. However, some services in DURAARK are depending on an internet connection (e.g. the semantic enrichment; the SIP upload to a digital preservation service) and will not be usable without it. Still, the session-based design of the Workbench allows to perform the steps where no internet connection is required and pass on the session to an internet-enabled computer to resume the session there.

---

[12]atom-shell: https://github.com/atom/atom-shell/
[13]node-webkit: https://github.com/rogerwang/node-webkit/

5 Licenses

The following table gives an overview of the software licences generated and used for the web services and UI modules implementation:

IPR Type	**IP used or generated**	**Software name**	**License**	**Information**
software	generated	DURAARK Framework	MIT	D2.4
software	generated	DURAARK Workbench	MIT	D2.4
software	used	Backbone.Marionette	MIT	http://marionettejs.com/
software	used	NodeJS	MIT	http://nodejs.org/

Licenses regarding the components from **D3.3** can be found in the respective report.

6 Conclusions & Impact

The "Workbench" as an integrated software prototype provides a platform for integrating existing and future software prototype deliverables into a set of workflows. Stakeholders defined in earlier reports are able to perform the use cases UC1, UC2, UC3, UC8 and UC9 when stepping through the four provided workflows *SIP Generation*, *Search & Retrieval*, *Geometric Enrichment* and *Semantic Archive Maintenance*.

The DURAARK Workbench is divided into two conceptual parts; the web services providing functionality in the context of long-term archival of BIM data, as well as a graphical user interface to access the functionality from the point of view of a stakeholder. This conceptual separation of concerns is a central aspect of the project's architecture. After the lifetime of the project the prototype should be usable by various stakeholders either as a frontend user or as a developer. With the component based architecture it is possible to customize the workflows to fit the various needs of different stakeholders. This flexibility will support the acceptance of the Workbench as a service platform.

Developing the web services "separated" from the user interface forces the development to focus on how a service is exposed to the external world through a reasonable API. Moving forward with the corresponding GUI at the same time tests the API and allows to enhance it on the go. The result is a stable, well-tested interface to long-term archival services together with a GUI that shows how to use those services "the right way".

This is the first version of the software prototype and the internal structure as well as the GUI will incrementally be improved and adopted to the needs defined by the evaluation activities in WP7. The general architecture and design decision, however, proved to be suited for the purpose of this deliverable, namely to provide an integrated platform with workflows to perform long-term archival use cases in the context of BIM data.

Appendices

1 Service Endpoints - RESTful API Description

In this section the RESTful API enpoints are listed. Examples for accessing the API are given, as well as the corresponding JSON responses.

Internally the Workbench is using a "session" system to that holds the input files and information on them. Each session has an ID, most of the provided services are working based on that session id. A prerequisite therefore is the existence of a "session". In M18 a session can only be created via the WorkbenchUI, see Section 2.1.1.1 on how to do that. Currently the system provides two predefined session with ID "**0**" and ID "**1**". For the following examples one of those IDs will be used. When creating new sessions via the GUI those sessions can be used, too. The session ID can be found on the Session Page.

1.1 Session Management

API Description

Queries the data for the available sessions. The example response is listing two available sessions with ID "0" and ID "1". The second example only lists data from session "0".

Example query and response

Query: http://workbench.duraark.eu/services/session
Response:


```
[{
    "id": 0,
    "label": "CCO_DTU-Building127",
    "files": [{
        "id": 0,
        "path": "./fixtures/repository/CCO_DTU-Building127_Arch_CONF.ifc",
        "name": "CCO_DTU-Building127_Arch_CONF.ifc",
        "type": "ifc",
        "size": "10.74 MB"
    }, {
        "id": 1,
        "path": "./fixtures/repository/CCO_DTU-Building127_Arch_CONF.e57",
        "name": "CCO_DTU-Building127_Arch_CONF.e57",
        "type": "e57",
        "size": "535.30 MB"
    }],
```

```
    "uuid": "390685d6-e055-4fc0-9133-7384dee45c5e"
}, {
    "id": 1,
    "label": "Empire State Building",
    "options": {
        "demo_mode": true
    },
    "files": [{
        "id": 0,
        "path": "./fixtures/repository/CCO_DTU-Building127_Arch_CONF.ifc",
        "name": "CCO_DTU-Building127_Arch_CONF.ifc",
        "type": "ifc",
        "size": "10.74 MB"
    }, {
        "id": 1,
        "path": "./fixtures/repository/CCO_DTU-Building127_Arch_CONF.e57",
        "name": "CCO_DTU-Building127_Arch_CONF.e57",
        "type": "e57",
        "size": "535.30 MB"
    }],
    "uuid": "0ffe055e-1360-47d4-a16c-026880c9eba5"
}]
```


Listing 1: Example response listing all available sessions.

Query: http://workbench.duraark.eu/services/session/0

Example response


```
{
    "id": 0,
    "label": "CCO_DTU-Building127",
    "files": [{
        "id": 0,
        "path": "./fixtures/repository/CCO_DTU-Building127_Arch_CONF.ifc",
        "name": "CCO_DTU-Building127_Arch_CONF.ifc",
        "type": "ifc",
        "size": "10.74 MB"
    }, {
        "id": 1,
        "path": "./fixtures/repository/CCO_DTU-Building127_Arch_CONF.e57",
        "name": "CCO_DTU-Building127_Arch_CONF.e57",
        "type": "e57",
        "size": "535.30 MB"
    }],
    "uuid": "0ffe055e-1360-47d4-a16c-026880c9eba5"
```

```
}
```

Listing 2: Example response listing session with ID "0".

1.2 File Identification

API Description

Queries the DROID file identification component to identify the E57 file of a given session. The example response shows status for the E57 file in session "0".

Example query and response

Query: http://workbench.duraark.eu/services/fileid/0

Example response

```
{
    "name": "CCO_DTU-Building127_Arch_CONF.e57",
    "format": "fmt/643",
    "valid": true,
    "formatString": "E57 (point cloud)"
}
```

Listing 3: Example response showing the status of the E57 file identification of session "0".

1.3 IFC Meta-data Extraction

API Description

Triggers the IFC metadata extractor component to query the metadata for the IFC file of a given session. The example response shows metadata for the IFC file in session "0" as an RDF Turtle string wrapped into a JSON response.

Example query and response

Query: http://workbench.duraark.eu/services/ifcm/0

Example response

```
{
    "rdf":
   "@prefix dct: .
    @prefix dbp-prop: .
    @prefix geo-pos: .
    @prefix xsd: .
    @prefix duraark: .
    @prefix qudt: .
    @prefix dbpedia-owl: .
    @prefix foaf: .
    @prefix dc: .

     duraark:object_identifier "2eD6iPVCPF0ADV8eYNtazn"^^xsd:string .
     foaf:name "DTU 127"^^xsd:string .
     dbp-prop:startDate "1970-01-01 01:00:00"^^xsd:date .
     dbpedia-owl:buildingStartYear "1970-01-01 01:00:00"^^xsd:date .
     duraark:length_unit "MILLIMETRE"^^xsd:string .
     duraark:authoring_tool "Autodesk Revit 2013 Autodesk Revit 2013
         2013"^^xsd:string .
     duraark:authoring_tool "Eindhoven University of Technology ifcspfrdfcat
         0.01a"^^xsd:string .
     foaf:based_near [ geo-pos:lat "55.68300000" ; geo-pos:lon "12.55000000" ] .
     duraark:floor_count "8"^^xsd:integer .
     duraark:room_count "55"^^xsd:integer .
     dbpedia-owl:address "Lyngby"^^xsd:string .
     dc:creator "Morten Jensen"^^xsd:string .
     duraark:enrichment_vocabulary "http://dbpedia.org/property"^^xsd:string .
     duraark:enrichment_vocabulary "http://sws.geonames.org"^^xsd:string .
     duraark:enrichment_vocabulary "http://vocab.getty.edu/aat"^^xsd:string .
     duraark:enrichment_vocabulary "http://vocab.getty.edu/ontology"^^xsd:string .
"
}
```

Listing 4: Example response listing the metadata of the IFC file in session "0" encoded in RDF Turtle and wrapped into JSON.

1.4 E57 Meta-data Extraction

API Description

Triggers the E57 metadata extractor component to query the metadata for the E57 file of a given session. The example response shows metadata for the E57 file in session "0" as an JSON response.

Example query and response

Query: http://workbench.duraark.eu/services/e57m/0

Example response


```
{
    "e57_metadata": {
        "guid": "{7E3C7C9C-EFCB-4F5A-9A6F-98A08F72FB1B}",
        "version_major": 1,
        "version_minor": 0,
        "creation_datetime": {
            "year": 2011,
            "month": 11,
            "day": 3,
            "hour": 19,
            "minute": 5,
            "seconds": 35.548999786376953
        },
        "coordinate_metadata": "undefined",
        "scan_count": 1,
        "image_count": 1,
        "scan_size": 1,
        "image_size": 1,
        "scans": [{
            "name": "parking000",
            "guid": "{F0B3C105-325B-4FC9-9E01-3130153F9800}",
            "original_guids": [],
            "description": "",
            "sensor_vendor": "",
            "sensor_model": "",
            "sensor_serial_number": "",
            "sensor_hardware_version": "",
            "sensor_software_version": "",
            "sensor_firmware_version": "",
            "temperature": 0,
            "relative_humidity": 3.4028234663852886e+038,
            "atmospheric_pressure": 3.4028234663852886e+038,
            "acquisition_start": {
                "year": 1980,
                "month": 1,
                "day": 6,
                "hour": 0,
                "minute": 0,
                "seconds": 0
            },
            "acquisition_end": {
                "year": 1980,
                "month": 1,
                "day": 6,
                "hour": 0,
                "minute": 0,
                "seconds": 0
            },
            "pose": {
```

```
        "rotation": {
            "w": 0.99996960774189081,
            "x": -0.0074585516927261801,
            "y": -0.0022701539983015365,
            "z": 0
        },
        "translation": {
            "x": 89.951072690000004,
            "y": 1.8420018,
            "z": 2e-008
        }
    },
    "index_bounds": {
        "row_minimum": 0,
        "row_maximum": 3470,
        "col_minimum": 0,
        "col_maximum": 8213,
        "return_minimum": 0,
        "return_maximum": 0
    },
    "cartesian_bounds": {
        "x_minimum": -68.432470999999993,
        "x_maximum": 57.134830999999998,
        "y_minimum": -59.897230999999998,
        "y_maximum": 70.512130999999997,
        "z_minimum": -2.0202709999999997,
        "z_maximum": 3.779801
    },
    "spherical_bounds": {
        "range_minimum": 1.6562939999999999,
        "range_maximum": 90.929899999999989,
        "elevation_minimum": -1.0909121353667537,
        "elevation_maximum": 1.5701933463079427,
        "azimuth_minimum": 0,
        "azimuth_maximum": -6.4112263142845904e-007
    },
    "intensity_limits": {
        "intensity_minimum": 0,
        "intensity_maximum": 1
    },
    "color_limits": {
        "color_red_minimum": 0,
        "color_red_maximum": 255,
        "color_green_minimum": 0,
        "color_green_maximum": 255,
        "color_blue_minimum": 0,
        "color_blue_maximum": 255
    },
    "point_fields": {
        "cartesian_x_field": true,
        "cartesian_y_field": true,
        "cartesian_z_field": true,
        "cartesian_invalid_state_field": true,
```

```
            "spherical_range_field": false,
            "spherical_azimuth_field": false,
            "spherical_elevation_field": false,
            "spherical_invalid_state_field": false,
            "point_range_minimum": -268.43545599999999,
            "point_range_maximum": 268.43545499999999,
            "point_range_scaled_integer": 9.9999999999999995e-007,
            "angle_minimum": 0,
            "angle_maximum": 0,
            "angle_scaled_integer": 0,
            "row_index_field": true,
            "row_index_maximum": 4095,
            "column_index_field": true,
            "column_index_maximum": 16383,
            "return_index_field": false,
            "return_count_field": false,
            "return_maximum": 0,
            "time_stamp_field": false,
            "is_Time_Stamp_Invalid_field": false,
            "time_Maximum": 0,
            "intensity_field": true,
            "is_intensity_invalid_field": false,
            "intensity_scaled_integer": 3.0518509475997192e-005,
            "color_red_field": true,
            "color_green_field": true,
            "color_blue_field": true,
            "is_color_invalid_field": false
        },
        "points_size": 27802731
    }],
    "images": [{
        "name": "parking000",
        "guid": "{76BD148C-D22A-4FE3-8CB2-0FB01F96698B}",
        "description": "",
        "representation": "spherical",
        "acquisition_datetime": {
            "year": 1980,
            "month": 1,
            "day": 6,
            "hour": 0,
            "minute": 0,
            "seconds": 0
        },
        "associated_data3D_guid": "{F0B3C105-325B-4FC9-9E01-3130153F9800}",
        "sensor_vendor": "",
        "sensor_model": "",
        "sensor_serial_number": "",
        "pose": {
            "rotation": {
                "w": 0.70283815264201144,
                "x": 0.077691052038131911,
                "y": 0.088163333920523737,
                "z": 0.70157669443617587
```

```
                },
                "translation": {
                    "x": 89.951072690000004,
                    "y": 1.8420018,
                    "z": 2e-008
                }
            },
            "visual_ref_representation": {
                "jpeg_image_size": 0,
                "png_image_size": 0,
                "image_mask_size": 0,
                "image_width": 0,
                "image_height": 0
            },
            "pinhole_representation": {
                "jpeg_image_size": 0,
                "png_image_size": 0,
                "image_mask_size": 0,
                "image_width": 0,
                "image_height": 0,
                "focal_length": 0,
                "pixel_width": 0,
                "pixel_height": 0,
                "principal_point_x": 0,
                "principal_point_y": 0
            },
            "spherical_representation": {
                "jpeg_image_size": 0,
                "png_image_size": 23551883,
                "image_mask_size": 0,
                "image_width": 8187,
                "image_height": 3471,
                "pixel_width": 0.00076745772193576565,
                "pixel_height": 0.0007666681778157584
            },
            "cylindrical_representation": {
                "jpeg_image_size": 0,
                "png_image_size": 0,
                "image_mask_size": 0,
                "image_width": 0,
                "image_height": 0,
                "pixel_width": 0,
                "pixel_height": 0,
                "radius": 0,
                "principal_point_y": 0
            }
        }]
    }
}
```


Listing 5: Example response showing the metadata of the E57 file in session "0".

1.5 Semantic Enrichment

API Description

Triggers the semantic enrichment component and lists data-sets found by the semantic enrichment component based on metadata found (and eventually edited by the stakeholder) in the IFC file. The example response shows (an excerpt) of the list of found data-sets for session "0".

Example query and response

Query: http://workbench.duraark.eu/services/semanticenrichment/0

Example response


```
[{
    "dataset_id": "28",
    "dataset_name": "enipedia",
    "resource_id": "184805",
    "resource_uri": "http://enipedia.tudelft.nl/data/EU-ETS/person/S%F8ren%20Holm",
    "property_uri": "http://enipedia.tudelft.nl/data/EU-ETS/city",
    "resource_value": "184805
        http://enipedia.tudelft.nl/data/EU-ETS/person/S%F8ren%20Holm
        http://enipedia.tudelft.nl/data/EU-ETS/city"
}, {
    "dataset_id": "28",
    "dataset_name": "enipedia",
    "resource_id": "238963",
    "resource_uri": "http://enipedia.tudelft.nl/wiki/Copenhagen_Hydro_Powerplant",
    "property_uri": "http://enipedia.tudelft.nl/wiki/Property:City",
    "resource_value": "238963
        http://enipedia.tudelft.nl/wiki/Copenhagen_Hydro_Powerplant
        http://enipedia.tudelft.nl/wiki/Property:City"
}, {
    "dataset_id": "28",
    "dataset_name": "enipedia",
    "resource_id": "238963",
    "resource_uri": "http://enipedia.tudelft.nl/wiki/Copenhagen_Hydro_Powerplant",
    "property_uri": "http://www.w3.org/2000/01/rdf-schema#label",
    "resource_value": "238963
        http://enipedia.tudelft.nl/wiki/Copenhagen_Hydro_Powerplant
        http://www.w3.org/2000/01/rdf-schema#label"
}, {
    "dataset_id": "28",
    "dataset_name": "enipedia",
    "resource_id": "238963",
```

```
        "resource_uri": "http://enipedia.tudelft.nl/wiki/Copenhagen_Hydro_Powerplant",
        "property_uri": "http://semantic-mediawiki.org/swivt/1.0#page",
        "resource_value": "238963
            http://enipedia.tudelft.nl/wiki/Copenhagen_Hydro_Powerplant
            http://semantic-mediawiki.org/swivt/1.0#page"
    }]
```

Listing 6: (Truncated) Example response listing data-sets found by the semantic enrichment component.

1.6 SIP Generator

API Description

Triggers the SIP Generator component. The example response shows the URL for downloading the generated SIP.

Example query and response

Query: http://workbench.duraark.eu/services/semanticenrichment/0

Example response

```
{
    "url": "a2844222-8d49-4734-a4b7-322c2ffa64fc.zip"
}
```

Listing 7: Example response containing the URL for downloading the generated SIP.

1.7 PROBADO3D - List

API Description

Lists the metadata to all previous SIP generation entries. The example response contains a single entry.

The *start* and *count* parameters can be used for pagination.

Example query and response

Query: https://ogo.cgv.tugraz.at/api/Models?start=0&count=1

Example response


```
{
    "sessionId": a2844222-8d49-4734-a4b7-322c2ffa64fc,
    "startIndex": 0,
    "count": 1,
    "totalResultCount": 19,
    "resultItems": [{
        "documentIdentifier": 1015,
        "description": "Test Ingestion",
        "title": "CCO_DTU-Building127_Arch_CONF",
        "creatorPersonId": 4,
        "geoLocation": "<?xml version="1.0" encoding="utf-8"?><Point
            xmlns="http://www.opengis.net/gml"><pos>50.94158
            6.958498</pos></Point>",
        "physicalAssets": null,
        "fileInfos": []
    }]
}
```


Listing 8: The example response lists the metadata for previous generated SIPs.

1.8 PROBADO3D - Fulltext Search

API Description

Allows to search the metadata of all previous SIP generation entries. The example response contains a single result.

The *start* and *count* parameters can be used for pagination.

Example query and response

Query: https://ogo.cgv.tugraz.at/api/Models?fulltextQuery="CCO"&start=0&count=1

Example response


```
{
    "sessionId": a2844222-8d49-4734-a4b7-322c2ffa64fc,
    "startIndex": 0,
    "count": 1,
    "totalResultCount": 19,
```

```
    "resultItems": [{
        "documentIdentifier": 1015,
        "description": "Test Ingestion",
        "title": "CCO_DTU-Building127_Arch_CONF",
        "creatorPersonId": 4,
        "geoLocation": "<?xml version="1.0" encoding="utf-8"?><Point
            xmlns="http://www.opengis.net/gml"><pos>50.94158
            6.958498</pos></Point>",
        "physicalAssets": null,
        "fileInfos": []
    }]
}
```


Listing 9: The example response lists the metadata for previous generated SIPs.

2 Representational State Transfer (REST) Principles

The following is a simplified description of a selection of REST principles, the authorative description can be found in Roy Fielding's excellent PhD thesis[14]

Every resource has an ID In DURAARK a simple example to explain this principle is to imagine an uploaded IFC file. The file gets the id '0' and is accessible by this ID from other services or an UI module. For the web there is the unified concept for IDs: the URI. URIs make up a global namespace, having the advantage that resources behind a REST service are always accessible via the same URI, which can be shared and bookmarked.

Interlinkage between resources Via hyperlinking it is possible to link from one resource to the other. The different resources do not have to be provided by the same service, they can be distributed.

Use of standard methods The data behind an URI is served via the HTTP application protocol (which in turn is based on the TCP transport protocol). HTTP provides standard methods for accessing and manipulating the data encoded in the URI, which are e.g. GET, POST, PUT or DELETE. For every resource those standard methods provide a clear semantic on what the programmer intends to do with the resource. For instance, calling the DELETE method on an URI clearly states that the resource should be deleted. The standardized concept of the URI and the standard methods provided by HTTP give a clear guidance even without extensive documentation on how to use a REST interface.

Resources have multiple representations When accessing a URI to retrieve data the client can specify via a HTTP header entry which data format he wants to retrieve. In DURAARK the default (and currently only) format is JSON, however, it is of course possible to implement the service to also support XML as the result encoding.

[14] RoyFielding'sdescriptionofREST:http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm