DELIVERABLE

Project Acronym: **E-ARK**

Grant Agreement Number: **620998**

Project Title: European Archival Records and Knowledge Preservation

---

DELIVERABLE DETAILS

DELIVERABLE REFERENCE NO.	D6.2
DELIVERABLE TITLE	Integrated Platform Reference Implementation
REVISION	1.0

AUTHOR(S)	
Name(s)	Organisation(s)
Rainer Schmidt, Sven Schlarb Björn Skog	Austrian Institute of Technology GmbH (AIT) ES Solutions (ESS)
REVIEWER(S)	
Name(s)	Organisation(s)
Thor Dekov Buur Kuldar Aas Janet Delve	Magenta ApS (MAG) The National Archives of Estonia (NAE) University of Brighton (UoB)

Project co-funded by the European Commission within the ICT Policy Support Programme		
Dissemination Level		
P	Public	X
C	Confidential, only for members of the Consortium and the Commission Services	

REVISION HISTORY AND STATEMENT OF ORIGINALITY

Submitted Revisions History

Revision No.	Date	Authors(s)	Organisation	Description
.01	08-09-15	R Schmidt	AIT	Initial document and structure
.02	12-10-15	R Schmidt	AIT	Backend Architecture and Implementation
.04	24-11-15	S Schlarb	AIT	E-ARK Web Implementation
.06	10-12-15	R Schmidt	AIT	Content completed
.07	14-12-15	R Schmidt	AIT	Sections harmonized
.08	21-12-15	R Schmidt	AIT	Revision for internal review
.09	25-01-16	R Schmidt	AIT	Incorporated review comments
1.0	26-01-16	J Delve	University of Brighton (UoB)	English language check

Statement of originality:
This deliverable contains original unpublished work except where clearly indicated otherwise. Acknowledgement of previously published material and of the work of others has been made through appropriate citation, quotation or both.

EXECUTIVE SUMMARY

The document describes a set of software components that have been developed in order to realize the E-ARK Integrated Platform Reference Implementation Prototype for storing, searching, and accessing E-ARK Information Packages on a scalable infrastructure. The E-ARK Integrated Platform Reference Implementation Prototype integrates (a) an Information Package creation and management system, (b) a repository for content search and access, and (c) a scalable storage and execution environment based on Apache Hadoop.

The deliverable focuses on the software components that have been individually developed and deployed on top of the core infrastructure components (Hadoop, Lily, SolR). This complements deliverable D6.1 which describes the set-up and configuration of the repository and indexing frameworks.

This document accompanies a number of software development results which have been developed in order to realize the Integrated Platform Reference Implementation Prototype. Besides the individual software components, the document provides an overview of the overall system architecture, the integration approach and utilized interfaces.

Table of Contents

EXECUTIVE SUMMARY 3

Table of Contents 4

1. Introduction 6

2. Backend System Architecture 7

3. HDFS Storage Service 10

3.1. Component Description 10

3.1.1. Implementation 10

3.1.2. Role and Functionality 10

3.2. Download and Installation 11

3.2.1. Building the Application 11

3.2.2. Starting the service using the embedded HTTP server. 11

3.3. Service Interfaces 12

3.3.1. Interface Description Document 12

3.3.2. Message Encoding 13

3.3.3. Uploading data to HDFS: 14

3.3.4. Searching for Files 15

3.3.5. Checking File Integrity (Fixity) 15

3.4. Further Documentation 15

4. Lily Ingest Component 15

4.1. Component Description 15

4.1.1. Implementation 15

4.1.2. Role and Functionality 16

4.2. Download and Installation 16

4.3. Parallel Information Extraction and Ingestion 16

4.4. Handling Classified Documents 17

4.5. Identification and Grouping of Records 18

5. Frontend System – Architecture and Integration 20

6. E-ARK Web Project 21

6.1. The E-ARK Web User Interface 22

6.2. Asynchronous and parallel package processing 26
6.3. Task and Workflow Definition 29
7. Concluding Remarks on Interoperability 29

1. Introduction

The document describes a set of software components that have been developed in order to realize the E-ARK Integrated Platform Reference Implementation Prototype (IPRIP) for ingesting, storing, searching, and accessing E-ARK Information Packages[1] on a scalable infrastructure.

A goal of WP6 is the development of a software architecture for an integrated platform that delivers the aforementioned functionalities based on an integrated system of software components. The E-ARK Integrated Platform Reference Implementation Prototype (IPRIP) provides a specific implementation of this architecture which also incorporates results from E-ARK work packages WP4 (Archival Information Package (AIP) creation) and WP5 (Dissemination Information Package (DIP) creation). The aim of the E-ARK IPRIP is the development of a prototype solution that can be easily demonstrated, downloaded, and extended by individual institutions, without requiring the deployment of enterprise software and/or the help professional support.

Thus, the E-ARK IPRIP can be used to extend existing archival systems (e.g. to improve search and access) but it can also be installed as a standalone system. The software packages comprising the IRPIP can be deployed as a standalone environment for creating Information Packages, carrying out a content-based full-text search, and for accessing the content of Information Packages on a per-file basis. The E-ARK IPRIP can also be deployed next to existing archiving systems in order to enhance available services, like for example finding aids (using the full text index) and order management (using the content repository).

This document provides the system documentation of a software prototype that is based on a stack of scalable backend technologies, which are mainly used for content search and access. The system is designed to scale from a single host to a cluster deployment by employing technologies like Apache Hadoop, Solr, and the Lily repository. The frontend components implement E-ARK workflows and graphical user interfaces for Information Package creation, staging, and access. The used technologies are aligned with the open-source software ESSArch[2] (especially EPP), which provides the reference implementation for a production environment supporting E-ARK concepts. ESSArch is a suite of archive software components representing a total archival workflow, compliant with the OAIS model. The IPRIP design allows one to directly integrate and/or replace different prototype components with ESSArch components. In addition, it is possible to extend existing systems with the search and access infrastructure provided by the prototype using a loose coupling strategy. Concrete examples for utilizing EPP with the E-ARK IPRIP are also briefly discussed in this deliverable.

The E-ARK IPRIP consists of a set of software components which are deployed on top of and in combination with existing software platforms. The project has been set up in a publically accessible demonstration environment which allows users to experiment with the system. A scalable cluster deployment (currently utilizing 36 CPU cores and 96 TB of disk

---

[1] As defined by the E-ARK Common Specification: http://www.eark-project.com/resources/specificationdocs/50-draftcommonspec-1

[2] https://github.com/ESSolutions

storage) has been set up using a research cluster at AIT for large-scale testing. Integration tests have been performed with a dedicated installation of ESSArch at ESS.

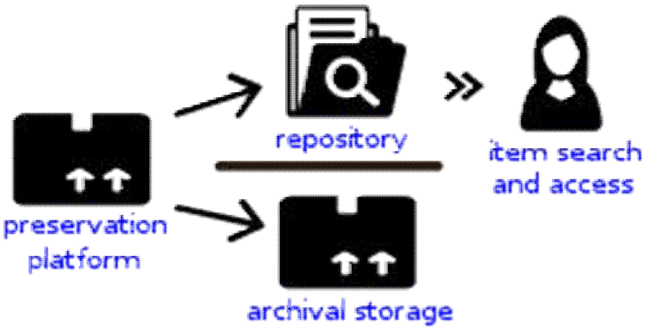


Figure 1*: Information packages are created, processed, and modified using an ingest/preservation platform like EPP or E-ARK Web. As part of the ingest workflow, the information packages are written to an archival storage medium. In addition, these packages are also ingested into an E-ARK content repository which provides advanced search and access services. The created repository records can be queried and accessed by an end-user application enabling random access on item/file level.*

* Icons made by Freepik

Figure 1 provides an architectural overview of the system as already introduced in deliverable D6.1. Deliverable 6.2 extends the indexing and search infrastructure described in D6.1 with a scalable repository providing file-level random access functionality. In addition, a frontend system has been developed that provides basic functionality for the specification, execution, and monitoring of ingest and access processes (for example AIP and DIP creation). A core-library encapsulates the functionality underlying the graphical user interface allowing one to easily integrate the prototype frontend (or parts of it) with existing information package management systems, like for example the ESSArch Preservation Platform (EPP), for handling the archival ingest workflow.

2. Backend System Architecture

The backend system of the IPRIP is built on top of an infrastructure that is based on the Apache Hadoop framework. The backend system can be deployed on a computer cluster allowing the repository infrastructure to scale-out. Horizontal scalability enables system administrators to increase the available system resources (i.e. storage and processing power) by adding new computer nodes. The number of nodes in a cluster is virtually unlimited and clusters may range from single node installations to clusters comprising thousands of computers.

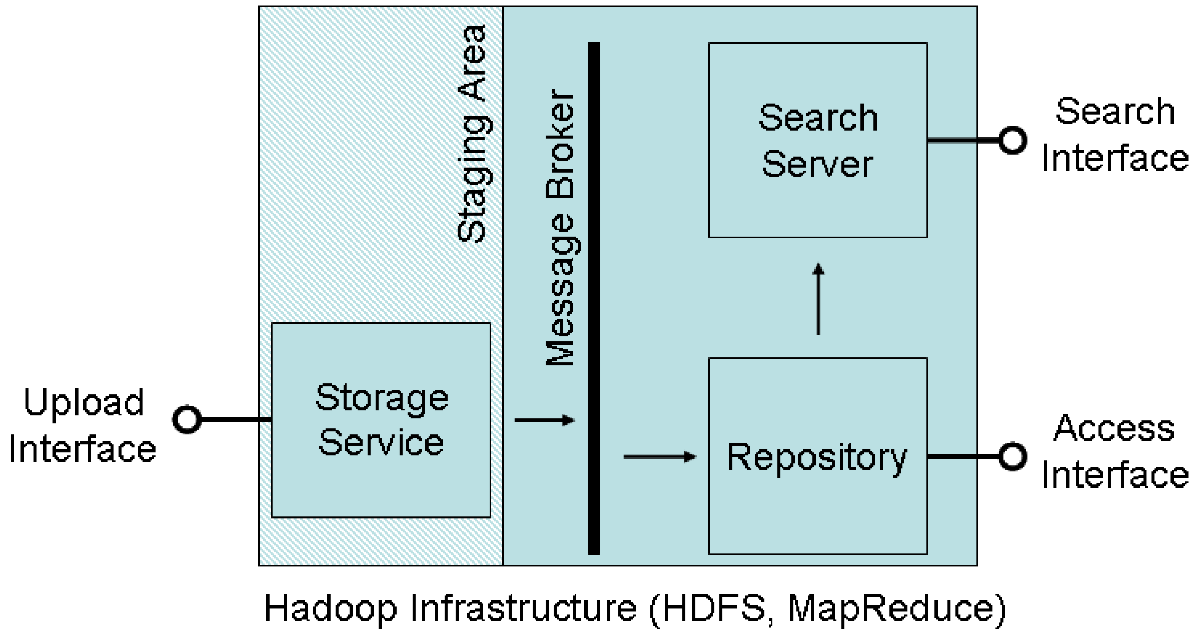


Figure 2: *System Components deployed on top a Hadoop infrastructure which is employed by the E-ARK Integrated Platform Reference Implementation Prototype. Hadoop can be deployed on a single node or on large clusters of commodity computers.*

The Hadoop infrastructure employed by the E-ARK IPRIP backend system provides a set of basic services. The most prominent ones are the distributed file system (HDFS[3]), the distributed database (HBase[4]), and the distributed execution system (MapReduce[5]). Figure 2 shows the main services provided by the prototype's backend system on top of the Hadoop cluster.

The major backend components are the HDFS Storage Service, the Repository, and the Search Server. These backend components make use of services like HDFS; HBase, and MapReduce, they provide external services to end-user applications, and interact with each other through internal communication mechanisms.

A Message Broker[6] is used to facilitate asynchronous communication between the components. The HDFS Storage service provides an HTTP interface allowing client applications (like for example an Information Package creation tool) to upload Information

---

[3] The Hadoop Distributed File System (HDFS) is the primary distributed storage used by Hadoop applications. http://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-hdfs/HdfsUserGuide.html
[4] Apache HBase is a distributed database (e.g. utilized by the Lily repository) that leverages the distributed data storage provided by HDFS. https://hbase.apache.org/
[5] Hadoop MapReduce is a software framework for writing applications which process large amounts of data on Hadoop clusters. https://hadoop.apache.org/docs/current/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html
[6] The IPRIP makes use of the RabbitMQ open source message broker software to trigger the ingestion of incoming data into the Repostitory. http://www.rabbitmq.com/

Packages onto the backend system's Staging Area, which is implemented atop the Distributed File System.

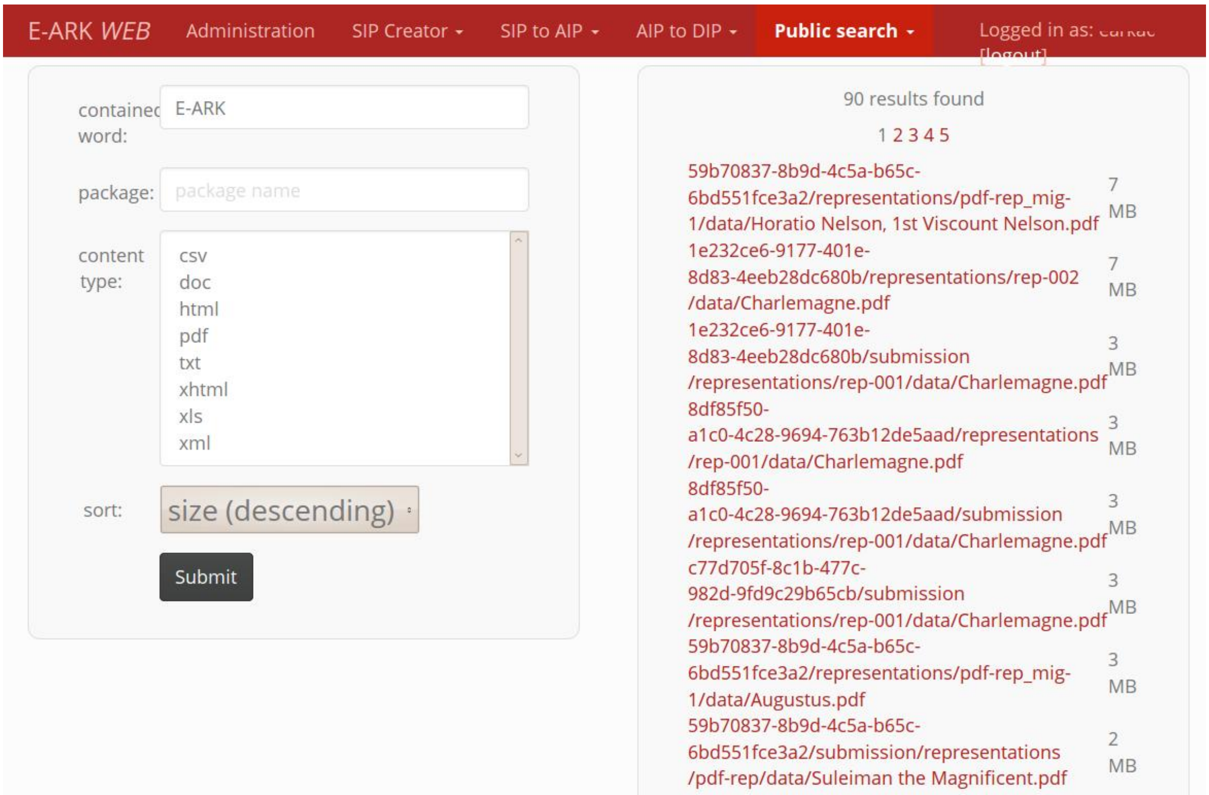
The Storage Service takes care of issues like the handling client requests, package identifiers, and file system I/O. Whenever new data has been uploaded, the Storage Service emits a message that can be used to trigger content ingestion[7] into the Repository.

The Repository is distributed across the Hadoop cluster and supports search, access, and storage methods for data items (i.e. files) that are contained in the uploaded Information Packages. The ingest procedure[6] creates repository records that contain the raw data together with extracted information (e.g. file name, textual content, file type) for the data items contained within an Information Package. The ingest procedure is triggered by messages received through the message broker. The Repository provides an HTTP REST interface that supports random access to data and metadata at item level based on unique URLs.

The Search Server maintains an index that supports searching over the records maintained by the Repository. The search index is updated whenever new records are available. The Search Server makes use of the Hadoop infrastructure to create and store its indexes. A HTTP-based Search Interface enables end-users and client applications to search and analyze the content of the Repository. Search results can be resolved using the Access Interface provided by the Repository. As an example, a full-text search dialogue is provided by the Reference Implementation's E-Ark Web UI, as shown in Figure 3.

---

[7] In this context, the term ingest does not refer to archival ingest and/or the creation of information packages but refers to the automatic creation of records for files contained in Information Packages and their storage in the content repository provided by the IPRIP.



contained word: E-ARK

package: package name

content type:

sort: size (descending)

90 results found

59b70837-8b9d-4c5a-b65c-6bd551fce3a2/representations/pdf-rep_mig-1/data/Horatio Nelson, 1st Viscount Nelson.pdf	7 MB
1e232ce6-9177-401e-8d83-4eeb28dc680b/representations/rep-002/data/Charlemagne.pdf	7 MB
1e232ce6-9177-401e-8d83-4eeb28dc680b/submission/representations/rep-001/data/Charlemagne.pdf	3 MB
8df85f50-a1c0-4c28-9694-763b12de5aad/representations/rep-001/data/Charlemagne.pdf	3 MB
8df85f50-a1c0-4c28-9694-763b12de5aad/submission/representations/rep-001/data/Charlemagne.pdf	3 MB
c77d705f-8c1b-477c-982d-9fd9c29b65cb/submission/representations/rep-001/data/Charlemagne.pdf	3 MB
59b70837-8b9d-4c5a-b65c-6bd551fce3a2/representations/pdf-rep_mig-1/data/Augustus.pdf	3 MB
59b70837-8b9d-4c5a-b65c-6bd551fce3a2/submission/representations/pdf-rep/data/Suleiman the Magnificent.pdf	2 MB

Figure 3*: A search dialog provided by the E-ARK Web graphical interface. Users may search content in and across Information Packages. Search results can be accessed directly via the returned HTTP URLs, which are pointing to the Content Repository's REST Interface.*

3. HDFS Storage Service

3.1. Component Description

3.1.1. Implementation

The Storage Service introduced in section 2 is implemented by the E-ARK HSink project which is available under Apache 2 licence as a public Github project[8].

HSink is a HTTP service for storing, searching, and retrieving potentially large files on a remote file system. Besides supporting standard OS file systems, HSink provides specific support for accessing the Hadoop Distributed File System (HDFS). The service is implemented using Apache Jersey 2.11 and uses an embedded Grizzly HttpServer if run in standalone mode. HSink utilizes HTTP chunked transfer encoding to support the transfer of large files. The service can be called using a standard HTTP client like *curl*. A Java client that invokes the service using the Jersey client API is dm-hdfs-storage-client[9].

3.1.2. Role and Functionality

---

[8] https://github.com/eark-project/dm-hdfs-storage

[9] https://github.com/eark-project/dm-hdfs-storage-client

The role of HSink in the IPRIP is the implementation of the Storage Service for depositing Information Packages on the HDFS Staging Area.

HSink provides a REST service for storing information Packets (e.g. in the form of tar-files) on a remote file system (e.g. HDFS). HSink handles the generation of unique URLs for uploaded packages allowing a user to identify an information Package independently of the utilized file names. In the current implementation, uploaded data cannot be altered or deleted by users. It is however possible to upload new versions of existing packages as many times as needed.

The URLs returned by HSink provide a handle for identifying and retrieving uploaded files on the Staging Area. HSink however does not make any assumptions on the identification mechanisms utilized by the Content Repository. Whenever a file is uploaded, HSink emits a message for the event which is accepted and forwarded by the message broker. The current implementation employs RabbitMQ for this purpose.

3.2. Download and Installation

3.2.1. Building the Application

The application depends on the E-ARK Maven projects global-configuration[10] and dm-parent[11]. Both projects must be downloaded, built, and installed before HSink can be compiled.

```
## download the project
git clone <THE_PROJECT_URL>
## move to the project's main directory
## build an install the project
mvn clean install
```

HSink can be easily built using Apache Maven. Please check Maven dependencies and versions in case the service is utilized in combination with HDFS.

```
git clone https://github.com/eark-project/dm-hdfs-storage.git
cd dm-hdfs-storage
mvn clean package
```

3.2.2. Starting the service using the embedded HTTP server.

HSink can be started in standalone mode based on an embedded Grizzly service using the command below.

```
java -jar target/hsink-service-1.0-SNAPSHOT-jar-with-dependencies.jar
```

---

[10] https://github.com/eark-project/global-configuration
[11] https://github.com/eark-project/dm-parent

Please note that the Maven dependency scope for Hadoop related libraries is set to <provided> by default. If HSink is used in combination with HDFS, it is therefore required to either include the required libraries with the Jar-File by changing the dependency scope to compile, or to launch HSink's main class org.eu.eark.hsink.Main with a Java classpath pointing to the required libraries of an Hadoop installation, as shown below. Please note that here the service is started in the background:

```
java -cp ./hsink-service-1.0-SNAPSHOT-jar-with-dependencies.jar:/usr/lib/hadoop/hadoop-common-2.0.0-cdh4.2.2.jar:/usr/lib/hadoop/lib/log4j-1.2.17.jar:/usr/lib/hadoop/lib/commons-configuration-1.6.jar:/usr/lib/hadoop/hadoop-auth-2.0.0-cdh4.2.2.jar:/usr/lib/hadoop/lib/slf4j-api-1.6.1.jar:/usr/lib/hadoop/lib/slf4j-log4j12-1.6.1.jar:/usr/lib/hadoop-hdfs/hadoop-hdfs-2.0.0-cdh4.2.2.jar org.eu.eark.hsink.Main filer=hdfsFiler fs.default.name=hdfs://localhost:8020 BASE_URI=http://demo.eark-project.com:8081/hsink/ > grizzly.out 2>&1 &
```

HSink is based on the Apache Jersey framework that supports the development of RESTful Web services. Jersey applications can be hosted using different deployment methods and environments. For deploying HSink in a servlet container, the reader is referred to the official Jersey documentation, section 4.7. Servlet-based Deployment[12].

3.3. Service Interfaces

HSink provides a set of HTTP interfaces for uploading, downloading, verifying, and discovering potentially large files on a remote file system like the IPRIP staging area.

3.3.1. Interface Description Document

At startup, the application should output a text similar to: "Jersey app started with WADL available at http://localhost:8081/hsink/application.wadl".

The Web Application Description Language (WADL) file provides an XML descriptor for HTTP-based services. The WADL file describes a REST service based on the resources it provides. In the case of HSink, the service has a default base path <protocol>://<server>:<port>/hsink providing one top level resource called *fileresource*.

The interface methods for retrieving/sending information from/to the server are described by a set of resources residing under the *fileresource* element. Using WADL, each method is described based on request and response messages including expected HTTP methods, parameters, and expected media types.

The following documents the REST API provided by HSink.

HTTP	Description of Messages
Resource:	/fileresource/files/{fileName}
Accepts:	[PUT] to creates new file object on storage. Content-type: application/octet-stream.

---

[12] https://jersey.java.net/documentation/2.11/deployment.html#deployment.servlet

Parameter:	String {filename} denotes the desired file name on the server.
Response:	201 Created. Location: /fileresource/files/{pathName}/{fileName}.
Resource:	/fileresource/files/{pathName}/{fileName}
Accepts:	[GET] to retrieve a file based on its URL.
Parameter:	String {pathName} generated file path, String {fileName} remote file name.
Response:	200 OK. Content-Type: application/octet-stream
Resource:	/fileresource/files/{pathName}/{fileName}/digest/{algorithm}
Parameter:	String {pathName} generated file path, String {fileName} remote file name. String {algorithm} the algorithm used to compute the digest - supported values are MD5, SHA-1, and SHA-256.
Response:	200 OK. Content-Type: text/plain
Resource:	/fileresource/files?name={fileName}
Accepts:	[GET] to retrieve a list of search results in the form of {pathName}/{fileName} that exactly match a given {fileName}.
Parameter:	String {fileName} remote file name.
Response:	200 OK. Content-Type: text/plain

3.3.2. Message Encoding

HSink makes use of chunked transfer encoding allowing a client to transmit large files using HTTP. The data transfer mechanism makes use of the Transfer-Encoding HTTP header and therefore requires a client that supports HTTP/1.1, like for example provided by the software package *curl*.

A Java client for HSink that makes use of the Jersey client API is available as a separate project on GitHub[13]:

```
usage: java -jar JARFILE [options] [source URI] [target URI]
...file upload:   java -jar client.jar -u ./file [http://localhost:8081/hsink]
...file download: java -jar client.jar -d http://localhost:8081/hsink/.../file ./file
...roundtrip test: java -jar client.jar -t ./file [http://localhost:8081/hsink]
```

---

[13] https://github.com/eark-project/dm-hdfs-storage-client

3.3.3. Uploading data to HDFS:

The curl example uploads a file called ./ip1234.tar to HSink. The specified remote filename is also set to ./ip1234.tar. Once the file has been received by HSink, the service responds with an URL for the generated resource which includes a generated location for the upload, as shown below. The generated URL can be used to download the file.

Example curl request:

```
curl -v -X PUT -H "Content-Type:application/octet-stream" -H "Transfer-Encoding: chunked"
http://localhost:8081/hsink/fileresource/files/ip1234.tar -T ./ ip1234.tar
```

Generated HTTP request:

```
PUT /hsink/fileresource/files/ ip1234.tar HTTP/1.1
User-Agent: curl/7.35.0
Host: localhost:8081
Accept: */*
Content-Type:application/octet-stream
Transfer-Encoding: chunked
Expect: 100-continue
```

Received HTTP response:

```
HTTP/1.1 100 Continue
HTTP/1.1 201 Created
Date: Tue, 30 Jun 2015 08:52:27 GMT
Location: http://localhost:8081/hsink/fileresource/files/2.2015-10-15/ip1234.tar
Content-Length: 0
```

3.3.4. Searching for Files

For each uploaded file HSink creates a new resource that can be identified and accessed based on a unique URL. The URL is returned as part of the HTTP response once the resource has been created (as shown in the above example).

HSink also provides a method to search for resource identifiers (i.e. remote location and file name) based on file names. When using this functionality, HSink returns a list of relative URIs identifying all resources whose file name exactly matches the search term.

Example curl request:

```
curl -v -X GET http://localhost:8081/hsink/fileresource/files?name=ip1234.tar
```

The functionality is utilized to check whether a file with the same name has already been uploaded to the staging area (and subsequently been ingested to the content repository)[14].

3.3.5. Checking File Integrity (Fixity)

HSink supports the creation of a checksum (digest) for each file that has been uploaded to the server. Supported hash functions to calculate the checksum are MD5, SHA-1, and SHA-256. The checksums are calculated on request and not persistently stored with the uploaded files. This functionality is used by the IPRIP frontend components in order to check the integrity of a file that has been uploaded to HDFS.

Example curl request:

```
curl -v -X GET curl -v -X GET http://localhs/4.2015-09-23/name=ip1234.tar/digest/md5
```

3.4. Further Documentation

For a more detailed description of the HDFS Storage service, a technical documentation of configuration parameters and the REST API specification, as well as the source code, the user is referred to the HSink source project site[15].

4. Lily Ingest Component

4.1. Component Description

4.1.1. Implementation

The Repository introduced in section 2 is realized based on the Lily framework which provides a distributed data repository that can be deployed on top of Apache Hadoop clusters. The cluster deployment and configuration of Lily has been described in deliverable D6.1. In the following, we describe File Ingest, a component that has been specifically developed for integrating the HDFS Storage Service with the Lily Repository.

---

[14] This is mainly important as in the present configuration of IPRIP the filename is part of the Lily record identifier and assumed to be unique (section 4.5).

[15] https://github.com/eark-project/dm-hdfs-storage

4.1.2. Role and Functionality

File Ingest is available as a service that automates the ingestion of Information Packages that have been uploaded onto the staging area (section 2) into the Content Repository. Ingestion refers to the creation of structured records from files contained in Information Packages and their storage in the Content Repository provided by the IPRIP.

Communication between the components is accomplished through the Message Broker employed by the IPRIP (c.f. Figure 2) which connects the File Ingest component and the HDFS storage service. The File Ingest component utilizes Hadoop-based mechanisms for creating and ingesting large numbers of records into the Lily repository in parallel on multiple computer nodes. The component generates a Lily repository record for each file (i.e. data item) contained within an Information Package. The record generation process is configurable allowing one to derive arbitrary information items from a particular data item which are (besides the raw binary data) included with the record. Examples of information items are text (extracted from documents), mime type, file size, selected elements of accompanying metadata (e.g. concerning data confidentiality), context information on the Information Package a file originates from, and the location of the file within that package. The Repository ingest is a completely automated process which is triggered as soon as data arrives on the Integrated Prototype's Staging Area.

4.2. Download and Installation

The E-ARK File Ingest project is available under Apache 2 license as a public Github project[16]. The File Ingest component has been developed in Java and can be easily downloaded and compiled using standard tools like Maven and Git.

Example for starting the File Ingest component:

```
hadoop jar target/dm-file-ingest-mapreduce-job.jar -libjars ${LIBJARS} -t eark1 -z localhost >
file-ingest.out 2>&1 &
```

In the example above, the option <libjars> is used to supply the Hadoop classpath ('hadoop classpath'), including HBase ('hbase mapredcp') and Lily client libraries ('lily-mapreduce-classpath'). The option -t is used to select the table name Lily should use for storing the records. The option -z is used to supply the zookeeper connection string which needs to be configured when setting up Lily. Optionally, one can execute the file ingest service in the background and redirect its logging output to a file (i.e. file-ingest.out), as shown in the example.

4.3. Parallel Information Extraction and Ingestion

The file ingestion component listens for messages on Information Packages that have been newly uploaded to the Staging Area. Incoming messages are buffered and the component triggers a new ingestion job depending on the frequency of incoming packages. This is done to foster the processing of multiple Information Packages as part of a single ingest procedure carried out on the cluster, and to prevent the creation of a large number of Hadoop jobs if multiple Information Packages are uploaded.

---

[16] https://github.com/eark-project/dm-file-ingest

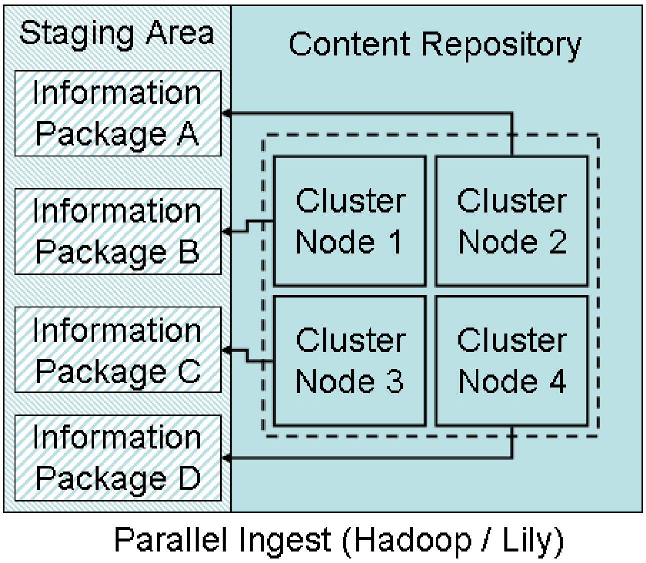


Figure 4*: Parallel Ingest of Information Packages into the distributed content repository on the cluster.*

Figure 4 provides a conceptual overview on how the parallel ingestion of uploaded Information Packages is handled. Once an ingest job is started, the available Information Packages are assigned the cluster nodes, which retrieve and process data items from the packages in parallel.

The file ingest component implements a (Hadoop-based) record reader that allows a node to read the data items (i.e. files contained within an Information Package) directly from the source packages (i.e. Information Packages stored on the HDFS storage area). As this process is implemented using data streams, files are extracted in memory and there is no need to write the extracted files to disk before they are ingested into the Content Repository.

The File Ingest component implements a configurable workflow for creating records that are ingested into the Content Repository. As the repository is distributed over the cluster providing a local access point on every node, the ingestion of the created records is also performed in parallel. As discussed earlier, records can include many different information items that have been retrieved from the original data item during the ingest workflow.

When records are ingested into the Content Repository, they are persisted within a distributed database which is Apache HBase in the case of the Integrated Prototype. Newly created or updated records are automatically indexed by the Search Server as described in deliverable D6.1.

4.4. Handling Classified Documents

Confidentiality of data items can be specified through metadata documents which are packaged and shipped within the Information Packages. The E-ARK AIP specification described in deliverable D4.3 provides information on data confidentiality as part of the preservation metadata. Here, data items - for example archived documents - can be classified using PREMIS documents which are contained within the Archival Information Packages (AIPs).

The handling of confidential data is a key concern when data is analyzed at scale and at a very fine granularity using automated processing techniques. Information extraction, automated structuring, and most importantly the creation of a search index are techniques that fall under this category. It is therefore essential that data confidentiality is respected when data is ingested into the Content Repository and exposed through the Search Server.

A couple of strategies can be implemented using the file ingest component to support the handling of restricted documents. In its current implementation, the file ingest component employs a field in the Lily repository record for specifying information related to data confidentiality. The field is then used to restrict search queries (and corresponding responses) to records that are not marked as confidential. The information on data confidentiality is retrieved from the PREMIS documents held by the ingested Information Packages. Documents are treated as confidential by default, e.g. in case confidentiality information is not provided.

The currently implemented strategy has the advantage that the visibility status of ingested documents can be easily changed based on a field that is part of the repository record. A disadvantage, however, is the fact that all ingested data items are added to the search index. Confidentiality with respect to search and access is implemented by the middleware (i.e. software) layer restricting the data that can be found and retrieved.

An alternative strategy that can be implemented using the Integrated Prototype’s file ingest component is to reject the ingestion of classified documents. Here, the ingest process is configured in a way that repository records are only created for unrestricted documents, while restricted documents are not being ingested into the Content Repository. These documents may still be archived as part of Archival Information Packages that are put on Archival Storage. However, they will not appear inside the Content Repository and the index of the Search Server at all. While this approach increases security by holding confidential data in the archival storage only, the drawback of the approach is that it is more cumbersome to react to documents that change their visibility status. In such cases, documents need to be removed from the index and/or be explicitly re-ingested and indexed.

Another alternative strategy would be to maintain an offline copy of the fully populated Content Repository database, and to physically remove all confidential records from the search index and (access) repository exposed to the user. The fully populated master copy of the Content Repository can then be utilized to generate new versions of the access repository database and search index that reflect the actual status of the records with respect to data confidentially. Here, the repository exposed to the user would (also physically) comprise only records that are not classified as confidential.

4.5. Identification and Grouping of Records

As described earlier, records created by the file ingest component are stored within Apache HBase, a distributed, schema free, NSQL database, utilized by the Lily repository. Hence, the data structures stored in the repository database reflect the records created by the file ingest component.

Although there is no pre-defined schema required by the database implementation, the structure of repository records must be defined for each index (e.g. corresponding to a collection) generated by the search server. The XML snippet shown in the table below provides an example configuration for a simple repository record suitable for Information Packages containing documents. Additional fields might be necessary for treating Information Packages containing richer content (e.g. multimedia content).

```
<fields>
        <field name="path" value="eark:path"/>
        <field name="contentType" value="eark:contentType"/>
        <field name="size" value="eark:size"/>
        <field name="confidential" value="eark:confidential"/>
        <field name="content" value="eark:content" extractContent="true"/>
</fields>
```

Here, a repository record contains fields defining a unique path, the content type (i.e. mime type information), the file size, a flag to identify files that are treated as confidential, and a field for containing the original content of the data item. In this example, there is no separate field specified for holding a text representation of the content, as this is automatically generated by the Search Server and added to the index.

A key design decision, however, is the design of the “path” field which is also used as the record identifier by the file ingest component. A “path” specifies the location of a particular data item within an Information Package. The example below provides a path element for a data item “Population-Export.xml” contained within the package “EXAMPLE_PACKAGE_141127”.

```
EXAMPLE_PACKAGE_141127/Content/Wildlife/Fox/Population-Export.xml
```

HBase is a tabular key/value store, providing random access to a database row given the identifier (or row key) is known. In the context of the IPRIP, this allows us to immediately retrieve records (i.e. database rows) for which the identifier (i.e. the path) is known. The value that can be retrieved for the identifier shown in the previous example would be the database record containing for example the content of the file “Population-Export.xml”, its mime-type, size, a timestamp, or other information items created as part of the ingest procedure.

Encoding the location of a data item within the record identifier is a deliberately made design decision. HBase stores records (or rows) based on lexicographically ordered row keys. This ordering scheme allows one to group related data items based on their identifier. Below, we extend the previous example showing a set of ordered identifiers.

```
EXAMPLE_PACKAGE_141127/IP_CS_mets.xml
EXAMPLE_PACKAGE_141127/Content/Wildlife/Fox/Population-Export.xml
EXAMPLE_PACKAGE_141127/Content/Wildlife/Fox/Population-Export.md5
EXAMPLE_PACKAGE_141127/Content/Wildlife/Deer/Population-Export.xml
EXAMPLE_PACKAGE_141127/Metadata/XYZ-Premis.xml
EXAMPLE_PACKAGE_163187/IP_CS_mets.xsd
```

The identifier set from the above example shows that although the file ingest component creates and ingests a separate record for each data item (i.e. a file) contained within an Information Package, the package structure is still maintained by the database using the natural ordering of the records in the database. The major benefit resulting from the ordering of records is the ability to process related data items without having to scan over the entire data collection. This capability is in particular important if one considers a large data collection containing millions of records. Ordered packages enable a user for example to browse the content of the entire Information Package, based on a single data item (or file) that might have been found by a user using the Search Server. Also, records resulting from data that is inserted or updated using various consecutive ingest steps are automatically ordered in the database based on the lexicographical ordering of the row identifier.

Other possible applications of this mechanism include the identification and aggregation of related data items along dimensions encoded in the row identifier (e.g. country/city/postal_code), which could be used to provide basic OLAP[17]-like operations. Providing support for analytical processing is however beyond the scope of the Integrated Protoype and not covered by this deliverable.

5. Frontend System – Architecture and Integration

In general, the backend system of the E-ARK Integrated Platform Reference Implementation Prototype takes Information Packages as input and provides functionalities like information extraction, search, and random access for the contained data items. In the previous chapters, we have outlined a set of custom components and services that which have been specifically developed to realize the IPRIP. The setup of the underlying search and access infrastructure (which is based on the Lily repository and the SolR search server) has been described Deliverable D6.1. The E-ARK Web Project described in section 6 provides a lightweight front-end implementation for this backend system.

The responsibility of the frontend system is the provisioning of user interfaces and corresponding services for creating information packages like AIPs and DIPs.

---

[17] Online Analytical Processing – see deliverable D4.3 for more on E-ARK and OLAP

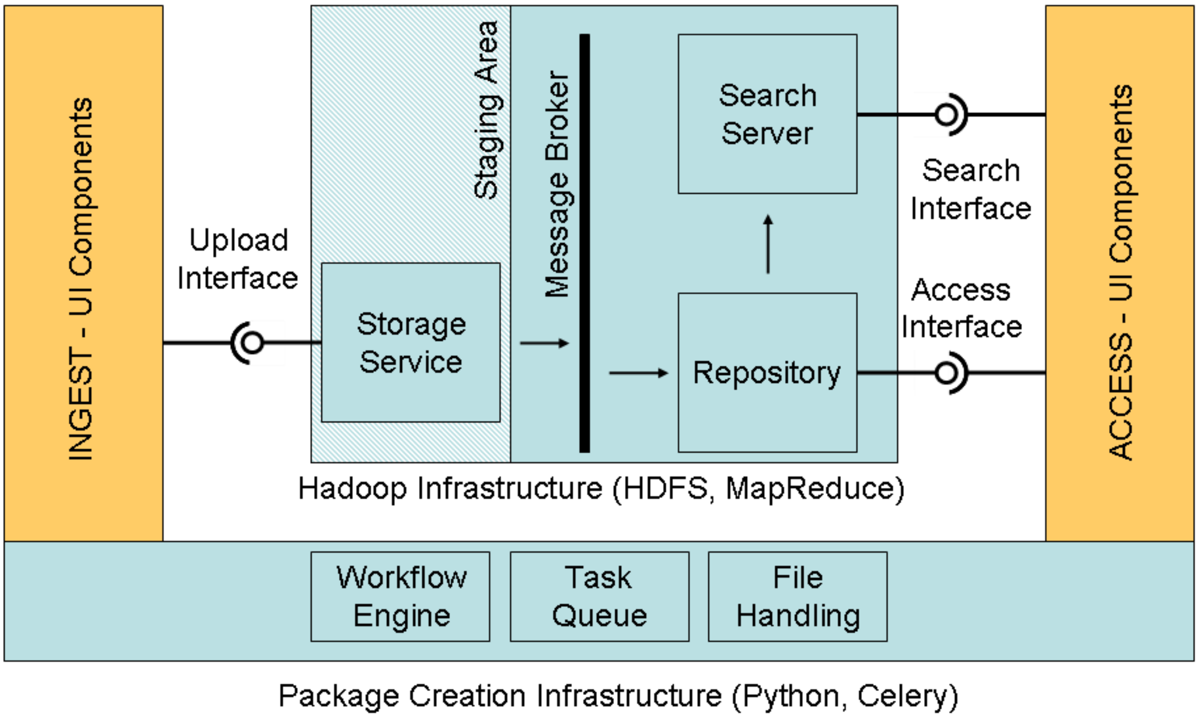


Figure 5: *System components of the Integrated Platform Reference Implementation architecture. The architecture consists of user interface components that support information package ingest and access processes. The frontend components are backed by a Package Creation Infrastructure handling file, task, and workflow processing. The frontend system is integrated with the Hadoop backend infrastructure for content extraction, storage, and search.*

Figure 5 provides an overview of the architecture implemented by the IPRIP consisting of a Package Creation Infrastructure implementing front end services (like package creation and user interfaces) and the Hadoop Infrastructure implementing the backend services (like indexing and access). As described in the previous sections, the frontend and the backend are loosely coupled based on REST services interactions. This design enables one to compose an integrated system using different backend and frontend implementations. The implementations provided by the IPRIP are light weight applications which are limited in their functionality and focused on distinct E-ARK principles.

The architecture implemented by the IPRIP is in general designed to support a loose coupling strategy so that existing systems can be combined with and/or be augmented with the IPRIP or particular components of the IPRIP platform.

6. E-ARK Web Project

The project *earkweb*[18] is a web application together with a task execution system which allows synchronous and asynchronous processing of Information Packages by means of processing units which are called “tasks”. The purpose of *earkweb* is, on the one hand, to provide a user interface for the IPRIP in order to showcase AIP transformation workflows which are being developed in the E-ARK project in an integrated way. On the other hand, the

---

[18] https://github.com/eark-project/earkweb

goal is to provide an architecture which allows reliable, asynchronous, and parallel creation and transformation of E-ARK Information Packages (E-ARK Submission Information Package (SIP), AIP, and DIP) integrated with E-ARK backend services for scalable and distributed search and access.

6.1. The E-ARK Web User Interface

The user interface of the E-ARK Integrated Platform Prototype Implementation Prototype is a Python[19]/Django[20]-based web application which allows managing the creation and transformation of E-ARK Information Packages (E-ARK IPs). It supports the complete AIP transformation pipeline, beginning with the creation of the SIP, over the conversion to an AIP, to the creation of the DIP which is used to disseminate digital objects to the requesting user.

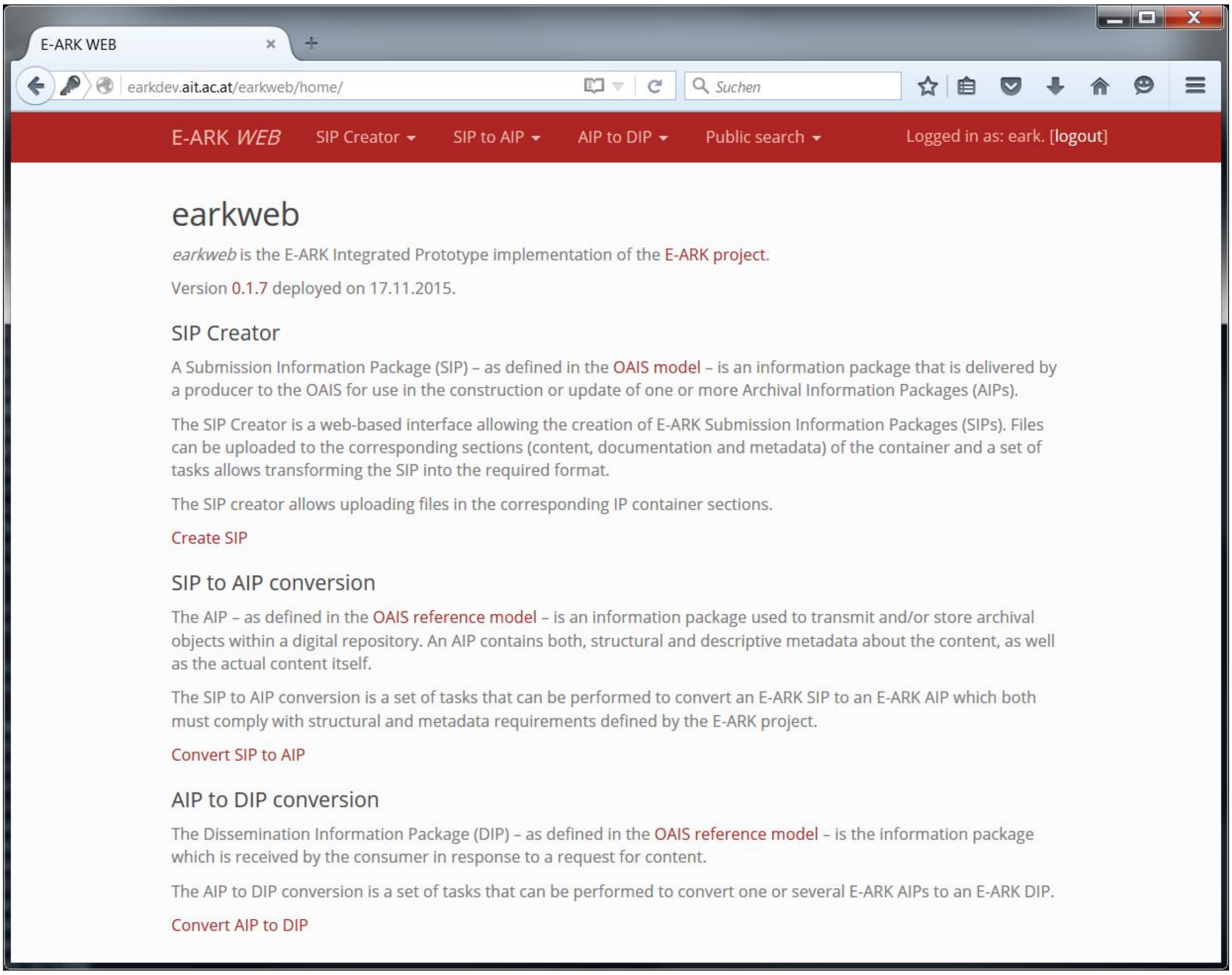
The *earkweb* user interface[21] is divided into four main areas: First, there is the “SIP creator” area which allows initiating a new SIP creation process and offers a set of transformation tasks to build E-ARK compliant SIPs. Second, there is the “SIP to AIP” area that allows the execution of tasks for converting an E-ARK compliant SIP to an AIP. Third, there is the “AIP to DIP” area which allows initiating a DIP creation process based on previously selected AIPs used for building the DIP with the help of a set of corresponding conversion tasks. And, finally, there is the “Public search” area offering full-text search based on the textual content available in the AIPs which have been uploaded to the HDFS staging area, ingested into Lily, and full-text indexed using SolR, as described in section 2. A screenshot of this user interface is shown in Figure 6.

---

[19] https://www.python.org

[20] https://www.djangoproject.com

[21] A demo instance of *earkweb* is available online (URL, user and password on can be provided by the E-ARK project office)



earkweb

earkweb is the E-ARK Integrated Prototype implementation of the E-ARK project.

Version 0.1.7 deployed on 17.11.2015.

SIP Creator

A Submission Information Package (SIP) – as defined in the OAIS model – is an information package that is delivered by a producer to the OAIS for use in the construction or update of one or more Archival Information Packages (AIPs).

The SIP Creator is a web-based interface allowing the creation of E-ARK Submission Information Packages (SIPs). Files can be uploaded to the corresponding sections (content, documentation and metadata) of the container and a set of tasks allows transforming the SIP into the required format.

The SIP creator allows uploading files in the corresponding IP container sections.

Create SIP

SIP to AIP conversion

The AIP – as defined in the OAIS reference model – is an information package used to transmit and/or store archival objects within a digital repository. An AIP contains both, structural and descriptive metadata about the content, as well as the actual content itself.

The SIP to AIP conversion is a set of tasks that can be performed to convert an E-ARK SIP to an E-ARK AIP which both must comply with structural and metadata requirements defined by the E-ARK project.

Convert SIP to AIP

AIP to DIP conversion

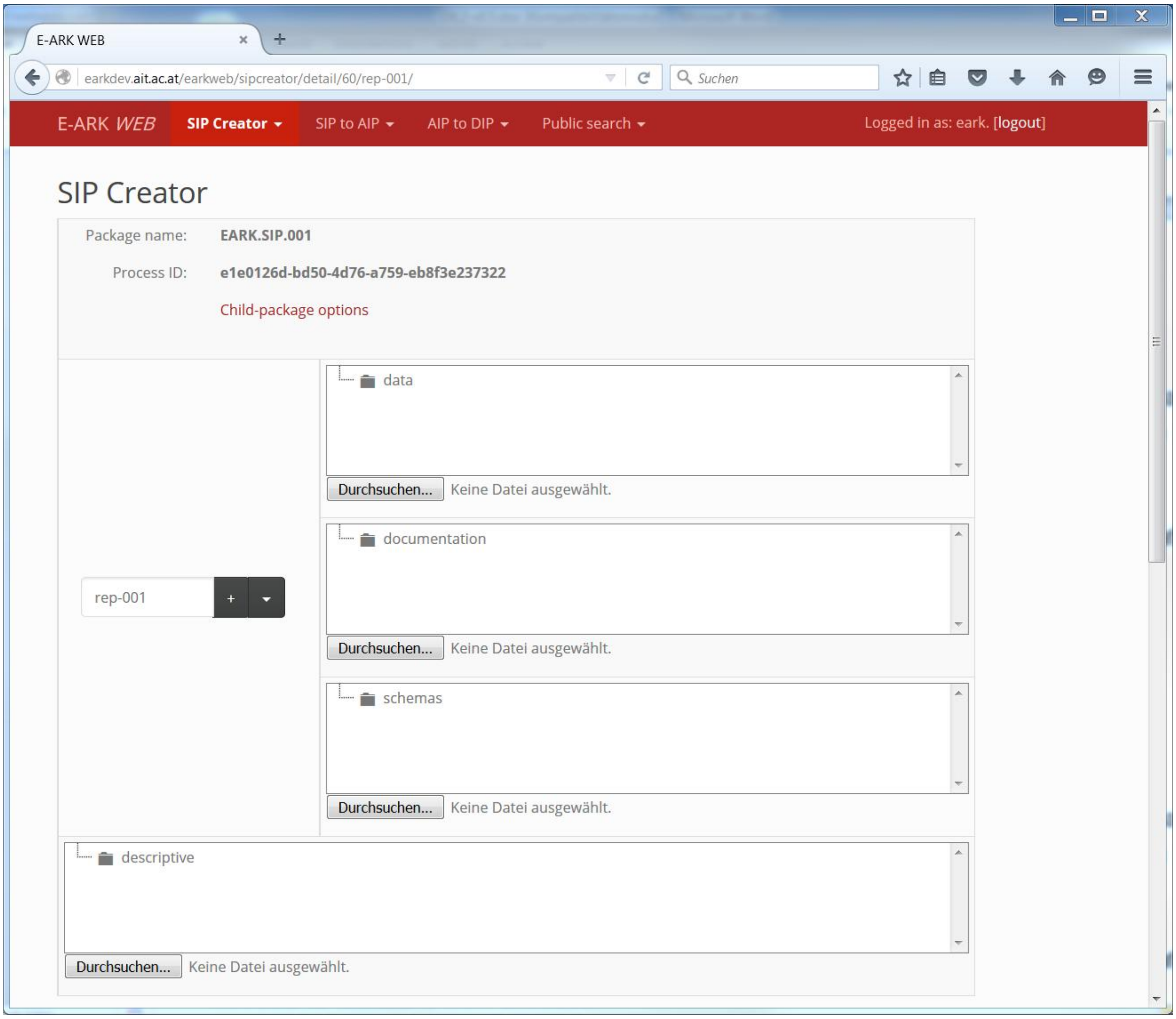
The Dissemination Information Package (DIP) – as defined in the OAIS reference model – is the information package which is received by the consumer in response to a request for content.

The AIP to DIP conversion is a set of tasks that can be performed to convert one or several E-ARK AIPs to an E-ARK DIP.

Convert AIP to DIP

Figure 6: *The earkweb user interface showing the four main areas “SIP creator”, “SIP to AIP”, “AIP to DIP”, and “Public search”.*

The common denominator of the “SIP creator”, “SIP to AIP”, and “AIP to DIP” areas is that they all offer Information Package transformation tasks. The transformation of Information Packages is implemented in the same way across all the three Information Package transformation areas. The “SIP creator” and the “AIP to DIP” areas additionally provide some basic setup forms in order to collect information needed to initiate a new process. As shown in Figure 7, the “SIP creator” provides a form which allows uploading individual files into the corresponding areas of the Information Package.



Package name: EARK.SIP.001

Process ID: e1e0126d-bd50-4d76-a759-eb8f3e237322

Keine Datei ausgewählt.

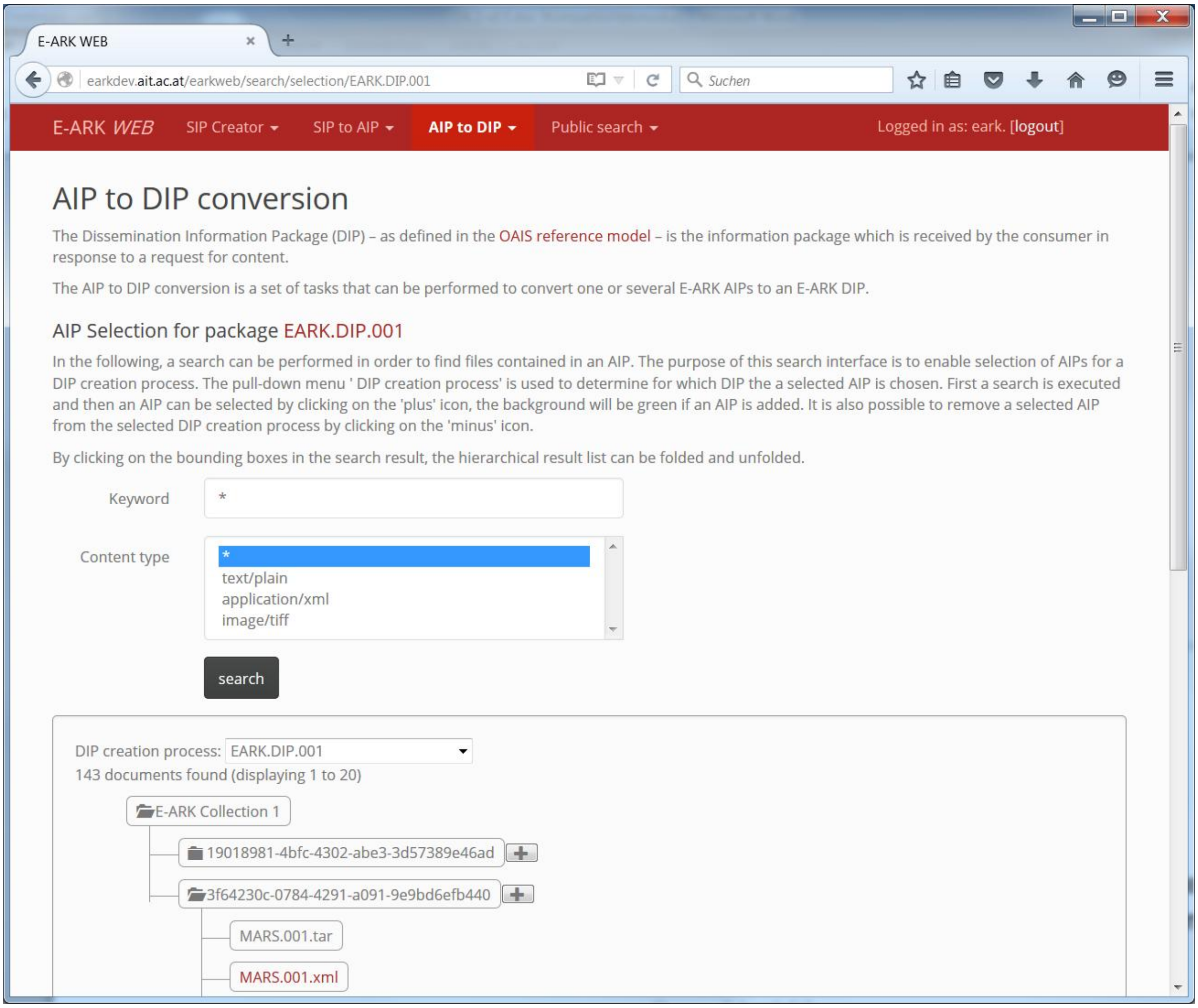
Keine Datei ausgewählt.

Keine Datei ausgewählt.

Keine Datei ausgewählt.

Figure 7: *User Interface of the “SIP creator” providing a form to upload individual files into the corresponding areas of the information package.*

Figure 8 shows a search interface used in the “AIP to DIP” dialogue that allows one to discover data in AIPs, select individual items, and generate DIPs.


E-ARK WEB
earkdev.ait.ac.at/earkweb/search/selection/EARK.DIP.001
Suchen
E-ARK WEB
SIP Creator
SIP to AIP
AIP to DIP
Public search
Logged in as: eark. [logout]
search

AIP to DIP conversion

The Dissemination Information Package (DIP) – as defined in the OAIS reference model – is the information package which is received by the consumer in response to a request for content.

The AIP to DIP conversion is a set of tasks that can be performed to convert one or several E-ARK AIPs to an E-ARK DIP.

AIP Selection for package EARK.DIP.001

In the following, a search can be performed in order to find files contained in an AIP. The purpose of this search interface is to enable selection of AIPs for a DIP creation process. The pull-down menu ' DIP creation process' is used to determine for which DIP the a selected AIP is chosen. First a search is executed and then an AIP can be selected by clicking on the 'plus' icon, the background will be green if an AIP is added. It is also possible to remove a selected AIP from the selected DIP creation process by clicking on the 'minus' icon.

By clicking on the bounding boxes in the search result, the hierarchical result list can be folded and unfolded.

Keyword: *

Content type: * (selected), text/plain, application/xml, image/tiff

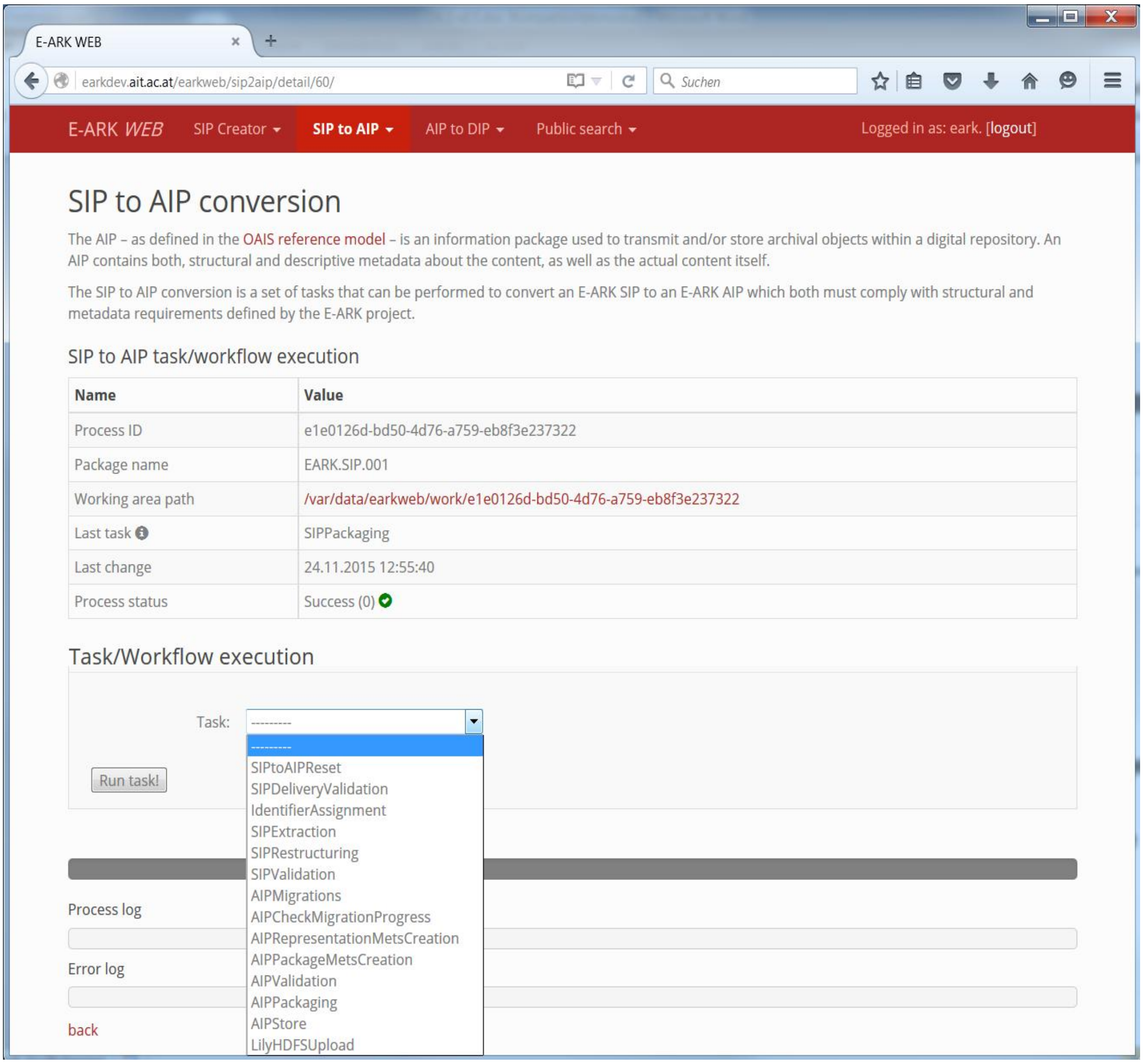
DIP creation process: EARK.DIP.001

143 documents found (displaying 1 to 20)

- E-ARK Collection 1
  - 19018981-4bfc-4302-abe3-3d57389e46ad
  - 3f64230c-0784-4291-a091-9e9bd6efb440
    - MARS.001.tar
    - MARS.001.xml

Figure 8: *Search interface in the “AIP to DIP” dialogue allowing a user to discover and select relevant data items from AIPs available in the Content Repository.*

The interface for executing tasks is basically the same across all package transformation areas. The difference lies in the tasks they provide. Figure 9 shows the task execution interface of the “SIP to AIP” conversion. The pull-down select field shows tasks that are available in this area. Here, the available tasks are related to Information Packages in a state between the initially submitted SIP and the AIP which is finally transmitted to the long-term storage and/or uploaded into the distributed storage area for full-text indexing and access.



SIP to AIP conversion

The AIP – as defined in the OAIS reference model – is an information package used to transmit and/or store archival objects within a digital repository. An AIP contains both, structural and descriptive metadata about the content, as well as the actual content itself.

The SIP to AIP conversion is a set of tasks that can be performed to convert an E-ARK SIP to an E-ARK AIP which both must comply with structural and metadata requirements defined by the E-ARK project.

SIP to AIP task/workflow execution

Name	Value
Process ID	e1e0126d-bd50-4d76-a759-eb8f3e237322
Package name	EARK.SIP.001
Working area path	/var/data/earkweb/work/e1e0126d-bd50-4d76-a759-eb8f3e237322
Last task	SIPPackaging
Last change	24.11.2015 12:55:40
Process status	Success (0) ✓

Task/Workflow execution

Task: ---------

Figure 9: *User interface for selecting and starting an information package transformation. This screenshot shows the SIP to AIP conversion area.*

6.2. Asynchronous and parallel package processing

As mentioned in section 6.1, the transformation of Information Packages is implemented in the same way across all of the three Information Package transformation areas. In this section, we describe the task execution infrastructure used by *earkweb* enabling the reliable and controlled execution of Information Package transformation tasks.

Apart from the Python/Django-based user interface, *earkweb* uses a backend for asynchronous and parallel task execution based on Celery[22], MySQL[23] databases, and RabbitMQ[24] as message broker software as shown in *Figure 10*.

---

[22] http://www.celeryproject.org
[23] https://www.mysql.com
[24] http://www.rabbitmq.com

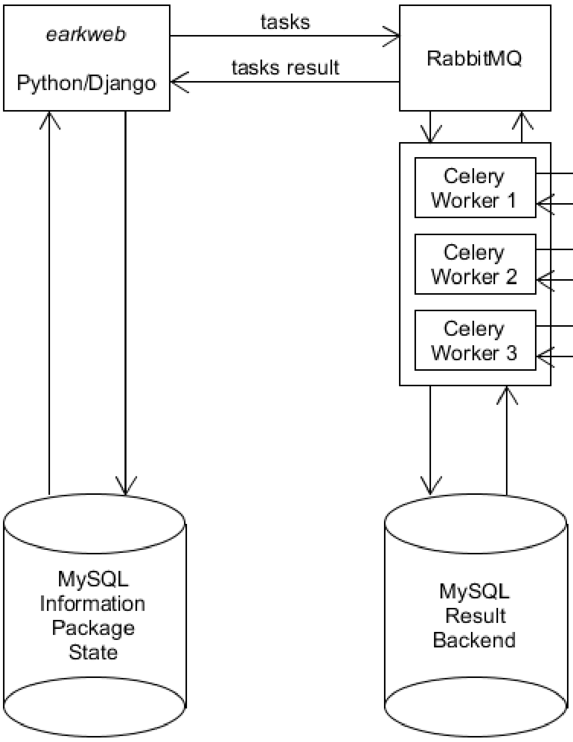


Figure 10: *Infrastructure for asynchronous and parallel task execution based on Celery, MySQL databases, and the RabbitMQ as message broker software backing the earkweb user interface components.*

Whenever a task is initiated using the *earkweb* task execution interface, the RabbitMQ message broker receives a message which is subsequently consumed by the Celery task execution engine. Tasks can be assigned to workers which are configured in the Celery backend. The workers share the same storage area. In Figure 10, this is realized using a NAS (Network Attached Storage) component.

The figure also shows two MySQL databases; The one on the left (Information Package State) is a database that supports a state machine, which is defined through Django model objects and basically translated into MySQL database tables. This database holds information such as the Information Package processing status, e.g. which was the last task executed, was the last task execution successful, time and date of task execution, etc. The database on the right (Result Backend) records the task execution results and is managed by Celery exclusively.

It is important to note that the purpose of the databases is only to provide information about the processing status to the user. The actual Information Package transformations do not depend on these databases. This means that the tasks do not have to be started by the user interface; the task can also be launched using a properly configured Celery client on the command line or even by directly using the

Python module containing the classes implementing the tasks.[25] The result of the package transformation is stored in the Information Package's working directory based on files (shown as "Working area path" in Figure 9).

As the actual status of the transformation process is persisted during task execution, it is not required to interrupt the processing chain for every execute task in order to update status information in the database. Based on the results stored in the working directory, the status of an Information Package transformation can be updated with a single operation when the transformation process has finished. This strategy increases the processing efficiency, which is critical when large volumes of data are processed, and helps avoide bottlenecks caused by a large number of parallel database connections.

Another advantage of this approach is that by design it is possible to reconstruct the databases tracking the status of the processed Information Package based on the information contained in the working directories. Particular importance was given to the principle of avoiding instantly recording digital object related processing information in the database as this may entail the risk of significantly increasing the processing time for very large Information Packages.

The decision to use either synchronous or asynchronous task execution for a specific task depends on the type of task and also the kind of data the Information Package contains. A task which itself initiates an unknown number of sub-tasks, can lead to a long task runtime, possibly beyond the defined timeout limit. An example would be a set of file format migration subtasks which are triggered for specific file types, e.g. each PDF file contained in an Information Package is converted to PDF/A. These cases can be implemented using a master task that starts an unknown number of sub-tasks and records the amount of migrations to be performed. This task is followed by a verification task which can be executed manually or automatically to report the current status of task executions. This way, it is possible to control that subsequent executions are not started before all sub-tasks were executed successfully, and that all the (possibly long-running) processes are decoupled from each other.

The upload of an AIP into the Hadoop infrastructure has been implemented as a synchronous task. The live progress of the upload process is shown directly in the user interface. However, if for cases where AIPs tend to be very large – where "large" is to be seen in the context of available bandwidth and read/write throughput – it is easily possible to change this task execution into an asynchronous task.

---

[25] https://github.com/eark-project/earkweb/blob/master/workers/tasks.py

6.3. Task and Workflow Definition

With respect to software design, a major goal was to foster flexibility, modularity, and extensibility of the task execution base class.[26] Tasks are implemented in one single Python script[27] and only contain the code that is necessary for the concrete task implementation. The intention is to keep the actual task implementation slim and offload extensive functionality into an *earkcore* python module[28] which can be made available to the Celery workers.

The E-ARK Web project defines a workflow model on top of the task execution layer. The "state" of an Information Package, as described earlier, is defined by storing the "last executed task" together with the success/failure of the execution. Tasks provide interface definitions (like for example "allowed inputs") which provide the basis for workflow composition. Using this information together with the current execution status, the workflow engine can control if a task is allowed to be performed on a specific Information Package.

New tasks can be easily added to the system by supplying a new task class implementation based on a Python script. The new task is available in the system as soon as the Celery workers are re-initialized. The configuration of the task is handled directly in the task implementation based on code annotations. Information to verify workflow composition is immediately available through the task description and does not require any additional configuration files. As the descriptive information is used to initialize the task configuration information in the database, it can be also dynamically adapted in the database, if required.

7. Concluding Remarks on Interoperability

The presented Integrated Platform Reference Implementation Prototype (IPRIP) combines a number of E-ARK outcomes including Information Package creation and transformation tools, the E-ARK Web frontend infrastructure, and the Hadoop-based backend infrastructure.

The reference implementation provides a light-weight prototype that can be accessed as an online demonstration environment, as open-source projects, and/or as downloadable packages. The IPRIP is however not meant to provide or contain a production environment. A major goal of the integrated prototype is to avoid dependencies to existing systems and/or complex environments that require professional support for their installation.

---

[26] https://github.com/eark-project/earkweb/blob/master/workers/default_task.py

[27] https://github.com/eark-project/earkweb/blob/master/workers/tasks.py

[28] https://github.com/eark-project/earkweb/tree/master/earkcore

The development of the integrated prototype has been closely aligned with the E-ARK tools and most notable ESSArch[29] which includes ESSArch Tools Producer (ETP), ESSArch Tools Archive (ETA) and ESSArch Preservation Platform (EPP). ESSArch, provided by ES Solutions, is for example in production use at the National Archives of Sweden and Norway. ESSArch is designed to be installed in an enterprise environment providing extended support for security, system integration, and the representation of preservation processes of archival institutions. ES Solutions also provides professional support for installing and customizing ESSArch.

In order to facilitate integration beyond the components that make up the IPRIP and to ensure the reusability of the produced outcomes, the IPRIP has been harmonized with ESSArch (especially EPP) with respect to core functionality, interfaces (task definition, HDFS upload), used programming language (Python and Python libraries), and underlying systems (Celery, Linux).

The HDFS storage service provided by the IPRIP provides a REST interface that can be utilized by existing ingest tools. This allows an ingest tool to store Information Packages besides the archival storage also on HDFS/Lily. Analogous to the upload task that has been implemented for the E-ARK Web Project, a “storage method” has been developed for EPP. This allows EPP to store Information Packages to HDFS (where they are subsequently ingested into Lily) in the same way packages are stored to archival storage systems like tape libraries.

As described earlier, a central part of the E-ARK Web project is the “E-ARK core” Python library that provides E-ARK specific functionality (e.g. metadata creation and validation), required to implement the Information Package creation and transformation processes. This library has been integrated in the latest EPP version providing a common code base for implementing E-ARK specific concepts in the package management system.

Specific tasks like for example package creation, validation, or transformation are implemented in E-ARK Web as Python functions relying on the E-Ark core library. This design allows EPP to efficiently reuse the code for the task implementation. As EPP as well as E-ARK Web rely on the Celary task processing system, the task execution is also facilitated in a technically aligned fashion.

The goal of the approach is to provide a light weight reference implementation that can be easily maintained, demonstrated, and handed over to individual users. By maintaining compliance with a production environment like EPP we verify the portability of the approach and address the long term sustainability of the results.

The IPRIP provides a full text index but does not employ its own archival catalogue as there are too many variations and different needs represented by the E-ARK

---

[29] https://github.com/ESSolutions

stakeholders. We assume that the IPRIP is deployed next to an already available catalogues like for example ICA-ATOM[30] or Blacklight[31].

---

[30] https://www.ica-atom.org/

[31] http://projectblacklight.org/