D.4.4.2 SIP-AIP conversion component: Reference Implementation

DOI: 10.5281/zenodo.1172925

Grant Agreement Number:	620998
Project Title:	European Archival Records and Knowledge Preservation
Release Date:	14th February 2018
Contributors	
Name	Affiliation
Sven Schlarb	Austrian Institute of Technology
Jan Rörden	Austrian Institute of Technology
Mihai Bartha	Austrian Institute of Technology
Kuldar Aas	National Archives of Estonia
Andrew Wilson	University of Brighton
David Anderson	University of Brighton
Janet Anderson	University of Brighton

STATEMENT OF ORIGINALITY

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This deliverable contains original unpublished work except where clearly indicated otherwise.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.

TABLE OF CONTENTS

List of Figures

List of Tables

Executive Summary

This deliverable D4.4 Part B is the second part of deliverable D4.4 and describes the SIP-AIP conversion component in its final version. The first part D4.4 Part A is an update of the previous deliverable D4.3 and represents the final version of the AIP specification. Figure 1 illustrates the two parts A and B of deliverable 4.4 and highlights Part B as the scope of the present document.

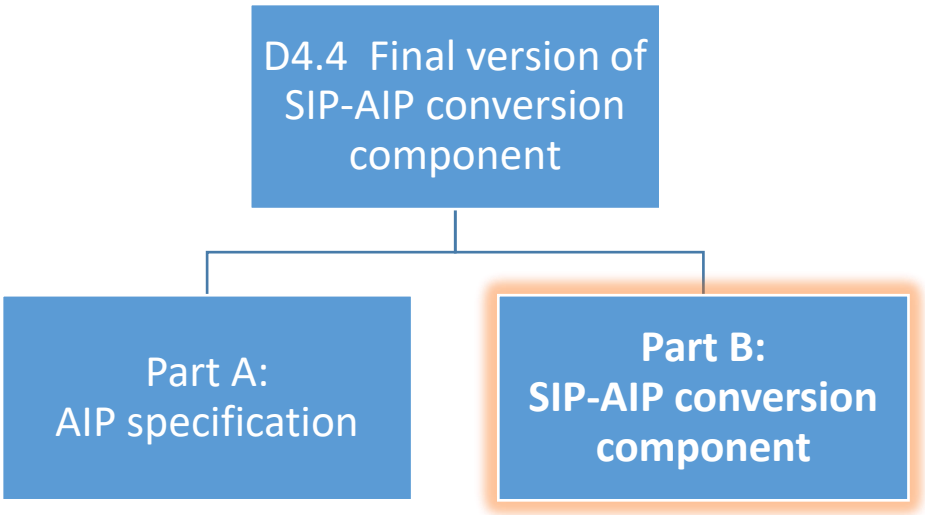


Figure 1: The scope of the this document is Deliverable D4.4 Part B

This document focuses on the E-ARK reference implementation of the SIP-AIP conversion component and describes the implementation of the basic concepts and definitions. The deliverable is a follow-up version of E-ARK deliverable D4.3 "E-ARK AIP pilot specification".[1]

The SIP-AIP conversion component is the reference implementation of the AIP format specification and is an integral part of the web application E-ARK Web (or in short: *earkweb)*.[2] The SIP-AIP conversion component consists of a set of individual tasks which are executed in a specific order to convert an E-ARK submission information package (SIP) into the E-ARK archival information package (AIP).

[1] http://eark-project.com/resources/project-deliverables/53-d43earkaipspec-1
[2] https://github.com/eark-project/earkweb; the actual SIP to AIP conversion component consists of a set of tasks in https://github.com/eark-project/earkweb/blob/master/workers/tasks.py.

1 Scope of this document

This document describes the SIP-AIP conversion component as part of the reference implementation E-ARK Web (in short: *earkweb.)*[3] It describes a set of tasks which is used to convert an E-ARK SIP to an E-ARK AIP and explains how these tasks are employed by the web application frontend of *earkweb*. In order to explain the context in which the SIP-AIP component is embedded, some details about the architecture and a user guide to the *earkweb* reference implementation are given.

2 Relation to other documents

This document describes how the SIP-AIP conversion is implemented as part of the component *earkweb*.[4] It complements part A "AIP format specification" of deliverable D4.4, "Final version of SIP-AIP conversion component".

Deliverable D4.4 has two previous iterations, Deliverable D4.2[5] and Deliverable D4.3.[6]

The SIP-AIP conversion component was designed to integrate with the scalable, Hadoop-based back-end implementation. The deliverable D6.3 "Data Mining Showcase" includes a detailed overview of the updated architecture of the Integrated Platform Reference Implementation (IPRIP). The SIP specification is described in the D3.x deliverables.

The fundamental document to understand the purpose of the SIP-AIP conversion is the Reference Model for an Open Archival Information System (OAIS).[7]

3 Introduction

earkweb is an open source archiving and digital preservation system. It is OAIS[8]-oriented which means that data ingest, archiving and dissemination functions operate on information packages bundling content and

[3] https://github.com/eark-project/earkweb; the actual SIP to AIP conversion component consists of a set of tasks in https://github.com/eark-project/earkweb/blob/master/workers/tasks.py.
[4] https://github.com/eark-project/earkweb; the actual SIP to AIP conversion component consists of a set of tasks in https://github.com/eark-project/earkweb/blob/master/workers/tasks.py.
[5] http://eark-project.com/resources/project-deliverables/18-d42-e-ark-aip-draft-specification
[6] http://eark-project.com/resources/project-deliverables/53-d43earkaipspec-1
[7] Reference Model for an Open Archival Information System (OAIS); Retrieved from http://public.ccsds.org/publications/archive/650x0m2.pdf, in the following we are always referring to this version 2 of the OAIS which was published in June 2012 by CCSDS as "magenta book" (ISO 14721:2012). The document is cited using the abbreviation OAIS.
[8] http://public.ccsds.org/publications/archive/650x0m2.pdf

metadata in contiguous containers. The information package format uses METS[9] to represent the structure and PREMIS[10] to record digital provenance information.

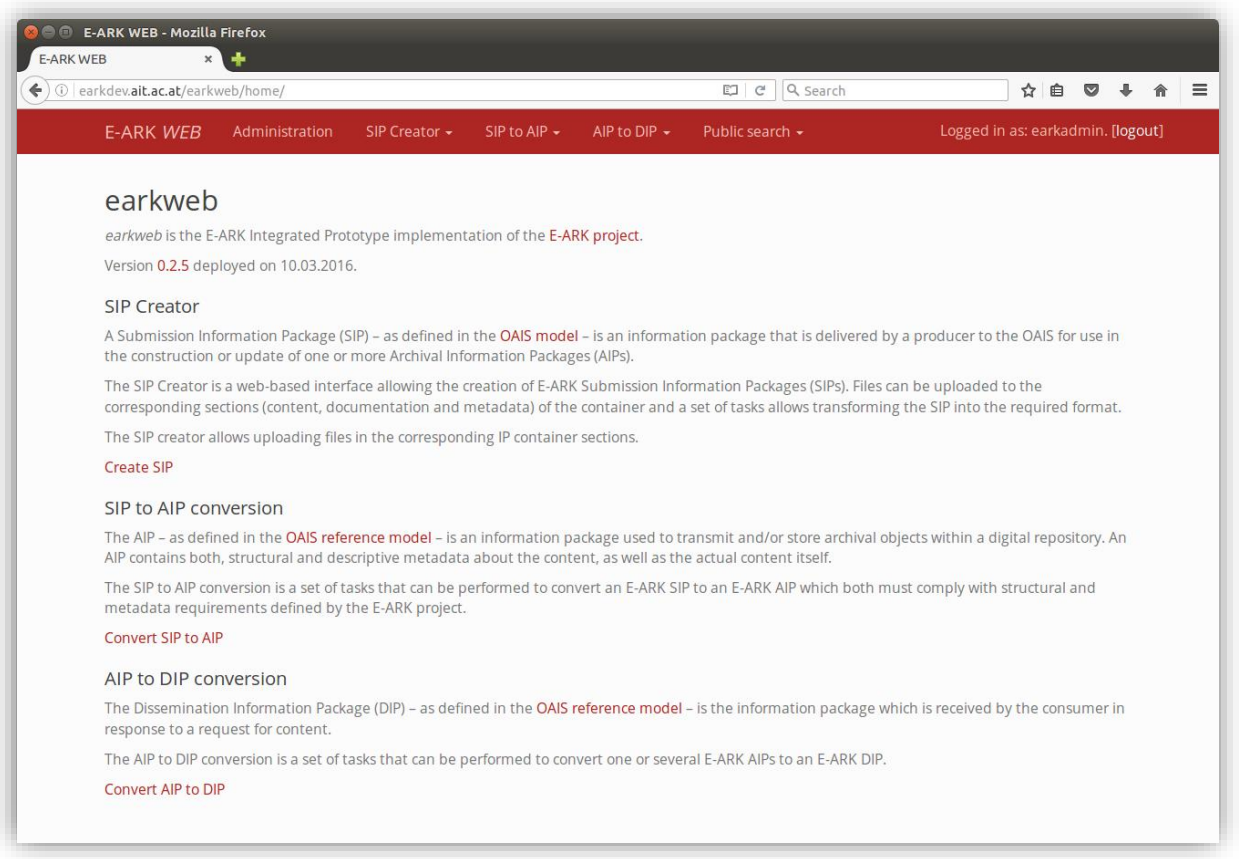


Figure 2: Screenshot of the initial page of the web application E-ARK Web

Earkweb offers functionality for the three types of information packages defined in the OAIS reference model: the Submission Information Package (SIP) which is the information sent from the producer to the archive, the Archival Information Package (AIP) which is the information stored by the archive, and the Dissemination Information Package (DIP) which is the information sent to a user when requested. The system allows executing different types of actions, such as information extraction, validation, or transformation operations, on information packages to support ingesting a SIP, archiving an AIP, and creating a DIP from a set of AIPs.

[9] http://www.loc.gov/standards/mets
[10] http://www.loc.gov/standards/premis

Earkweb consists of a frontend web application together with a task execution system based on Celery[11] which allows synchronous and asynchronous processing of information packages by means of processing units which are called “tasks”.

The backend can also be controlled via remote command execution without using the web frontend. The outcomes of operations performed by a task are stored immediately so that the status information in the frontend's database can be updated afterwards.

The SIP to AIP conversion component is implemented as a set of backend Celery tasks. These tasks can be executed using the web application frontend, by invoking the tasks in headless mode. *Earkweb* also offers a pre-defined workflow for batch processing which executes the full chain of tasks for automatic SIP-AIP conversion. These alternatives will be explained in detail throughout this document.

4 E-ARK Web Architecture overview

4.1 Cluster software stack

The E-ARK Web architecture is designed for efficiently processing, storing, and accessing very large data collections in terms of scalability, reliability, and cost. The system makes use of technologies like the Apache Hadoop[12] framework, NGDATA's Lily[13] repository, and the Apache SolR[14] search server allowing the repository infrastructure to scale-out horizontally. Using Hadoop, the number of nodes in a cluster is virtually unlimited and clusters may range from single node installations to clusters comprising thousands of computers. The diagram in Figure 3 gives an overview about this architecture.

[11] http://www.celeryproject.org/
[12] http://hadoop.apache.org/
[13] https://github.com/NGDATA/lilyproject
[14] http://lucene.apache.org/solr/

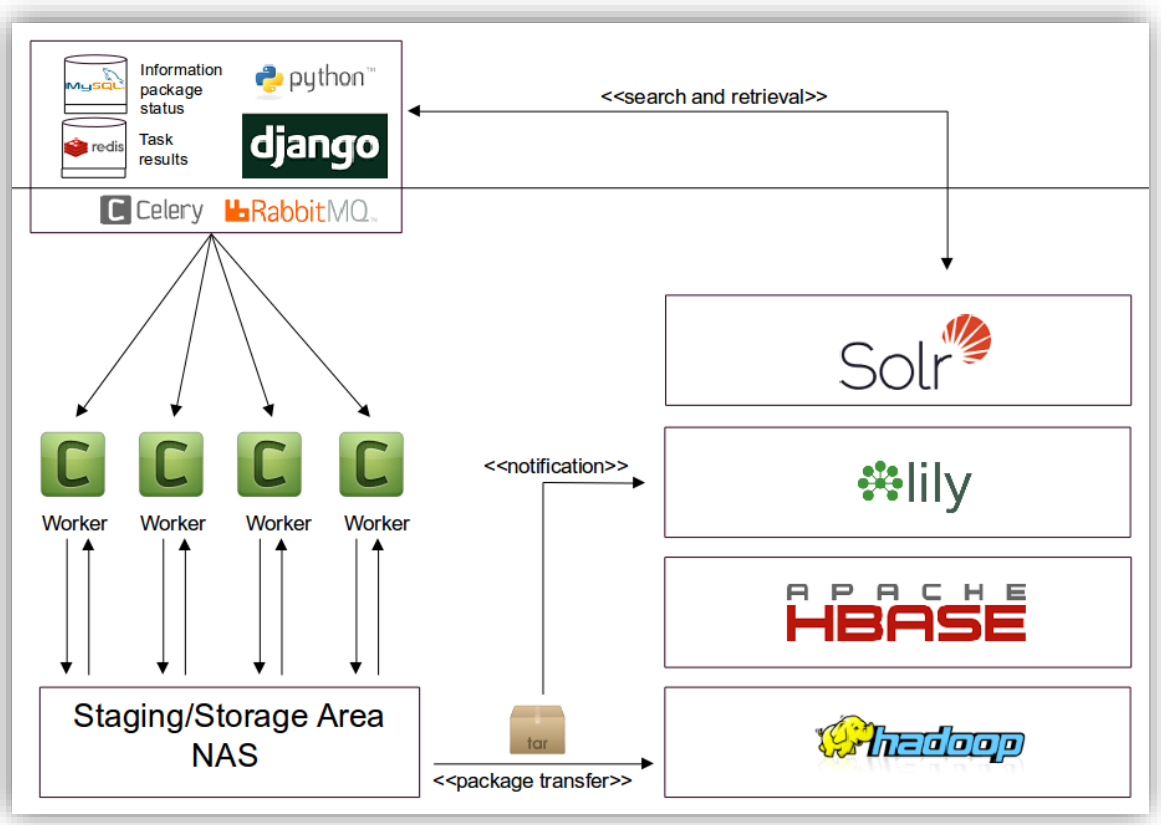


Figure 3: Architecture overview of the full-scale version of E-ARK Web

The user interface, represented by the box on top of the diagram, is a Python/Django-based web application which allows managing the creation and transformation of information packages. It supports the complete archival package transformation pipeline, beginning with the creation of the Submission Information Package (SIP), to the conversion to an Archival Information Package (AIP), to the creation of the Dissemination Information Package (DIP) which is used to disseminate digital objects to the requesting user. Tasks can be assigned to Celery workers (green boxes with a "C") which share the same storage area and the result of the package transformation is stored in the working directory.

Once the creation of information packages is finished, they can be deployed to the Lily access repository. Lily is built on top of HBase,[15] a NoSQL database that is running on top of Hadoop. Lily defines some data types most of which are based on existing Java data types. Lily records are defined using these data types rather

[15] https://hbase.apache.org/

than using plain HBase tables, which makes them better suited for indexing due to a richer data model. The Lily Indexer is the component which sends the data to the Solr server and keeps the index synchronized with the Lily repository. Solr neither reads data from HDFS[16] nor writes data to HDFS. The index is stored on the local file system and optionally distributed over multiple cluster nodes if index sharding (mechanism used to spread load on databases across multiple servers) or replication is used.

4.2 Standalone software stack

There is also a lightweight version of E-ARK Web where the large-scale storage backend (HDFS, HBase) is replaced by a conventional file system storage and the SolR search server is a single instance of SolR instead of a SolR Cloud deployment, as illustrated by the diagram in Figure 4.

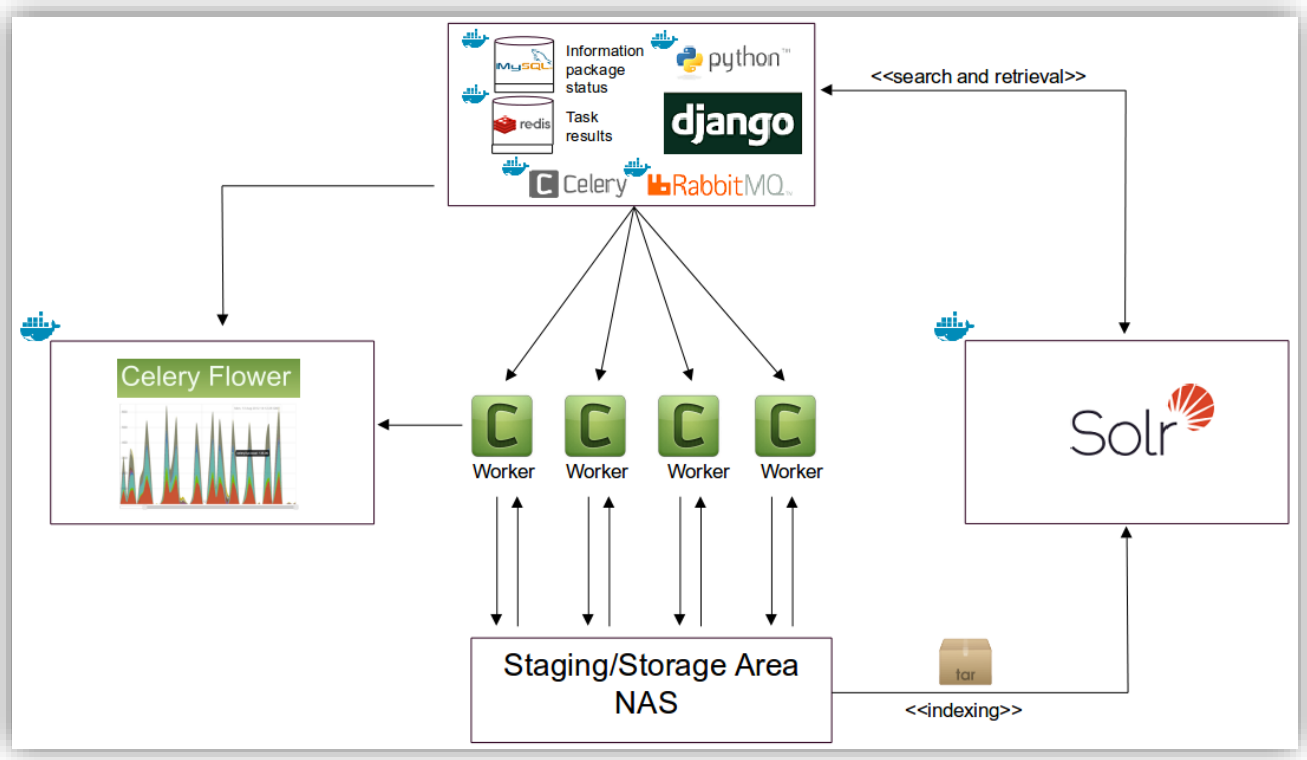


Figure 4: Architecture overview of the lightweight version of E-ARK Web

The standalone version of the IPRIP e-archiving environment has been made available using a container-based

[16] „Hadoop Distributed File System (HDFS) is designed to reliably store very large files across machines in a large cluster.", https://wiki.apache.org/hadoop/HDFS

deployment model based on Docker[17] in order to support simple and modular installation of the software.

Docker is an open-source engine that automates the deployment of any application as a lightweight and portable container that will run on any platform where the Docker engine is supported.[18] In order to allow deploying IPRIP services on a Docker platform, Docker containers for the individual services of *earkweb*'s frontend and backend have been created.

Table 1 shows the software module and the corresponding image which is used in a Docker deployment. The left column designates the software module and the right column indicates the image which is used to create and run a container to provide the corresponding service.

Software module	**Docker image**
MysSQL[19]	earkdbimg[20]
SolR[21]	Solr[22]
RabbitMQ[23]	tutum/rabbitmq[24]
Redis[25]	tutum/redis[26]
E-ARK Web[27]	earkwebimg[28]
Celery[29]	earkwebimg
Celery Flower[30]	earkwebimg

Table 1 Docker containers of the standalone deployment stack

Docker Compose[31] is used to run the components listed in Table 1 as multi-container Docker applications. Docker Compose allows automatically retrieving or building required images and containers as well as controlling start up and shut down of multiple inter-dependent services. Docker Compose also allows the linking of services to make inter-service communication easier. Figure 5 shows the YAML file which defines

[17] https://www.docker.com
[18] https://docs.docker.com/engine/installation
[19] http://www.mysql.com
[20] Based on earkweb image created from http://github.com/eark-project/earkweb
[21] https://lucene.apache.org/solr
[22] https://hub.docker.com/_/solr
[23] http://www.rabbitmq.com
[24] https://hub.docker.com/r/tutum/rabbitmq
[25] http://redis.io
[26] https://github.com/tutumcloud/redis
[27] http://github.com/eark-project/earkweb
[28] https://github.com/eark-project/earkweb/blob/master/Dockerfile
[29] http://www.celeryproject.org
[30] https://github.com/mher/flower
[31] https://github.com/docker/compose

the service composition so the services can be run together in an isolated environment. The dotted lines are used to visually divide the YAML file according to the different services it defines.

To give one example, the first service, named 'db', provides the MySQL database service needed by the *earkweb* frontend web application to store basic information about the processing status of information packages. The service is defined by the following properties:

- **image**: earkdbimg – The name of the image which is used to provide the MySQL database.
- **container_name**: earkdb_1 – The name of the Docker container.
- **build**: ./docker/earkdb – The Docker file which contains the build instructions to create the Docker image.
- **volumes**: - /tmp/earkweb-mysql-data:/var/lib/mysql – A directory from the Host system (here: /tmp/earkweb-mysql-data) is mounted as the MySQL data directory into the Docker container (here: /var/lib/mysql).
- **ports**: - '3306:3306' – The Port of the application is mapped to the port where the service will be exposed (in this case the standard MySQL port 3306 will be exposed as port 3306 by the container).

Furthermore, the orange arrows in Figure 5 show how the services are linked to each other using the "links" attribute of the *earkweb* container to reference the required service by name or the BROKER_URL environment variable of the flower service to link to the 'rabbitmq' service.

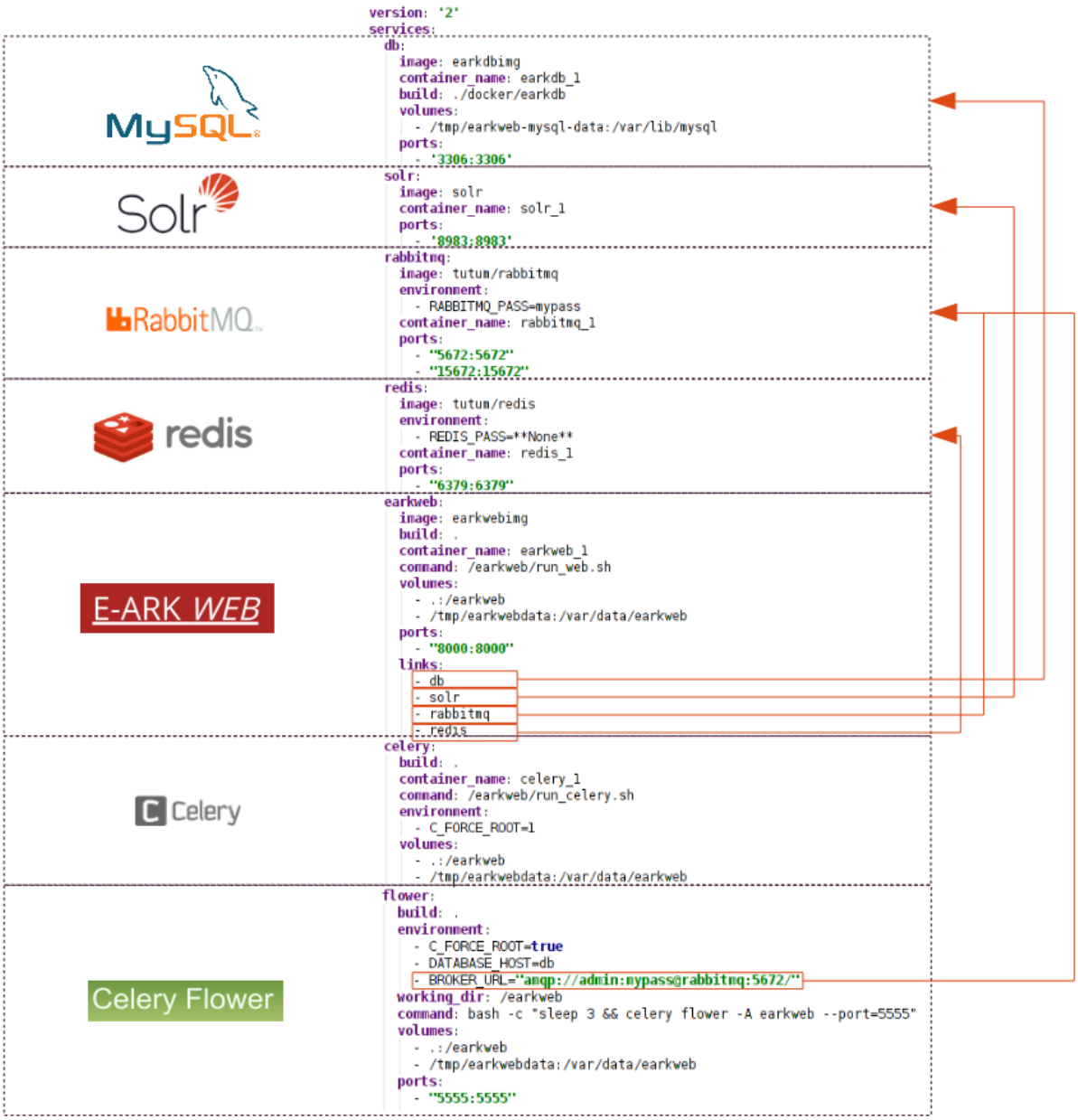


Figure 5 Docker containers of the Standalone Software Stack and the main links between the containers.

Figure 6 shows the origin of the images which are needed to run the various Docker containers. It shows that the Solr, RabbitMQ, and Redis images are directly retrieved from the Docker Image Library[32], and that the

[32] https://hub.docker.com

other containers are based on Docker container build instructions provided in the form of Dockerfiles[33] as part of the *earkweb* code base.[34]

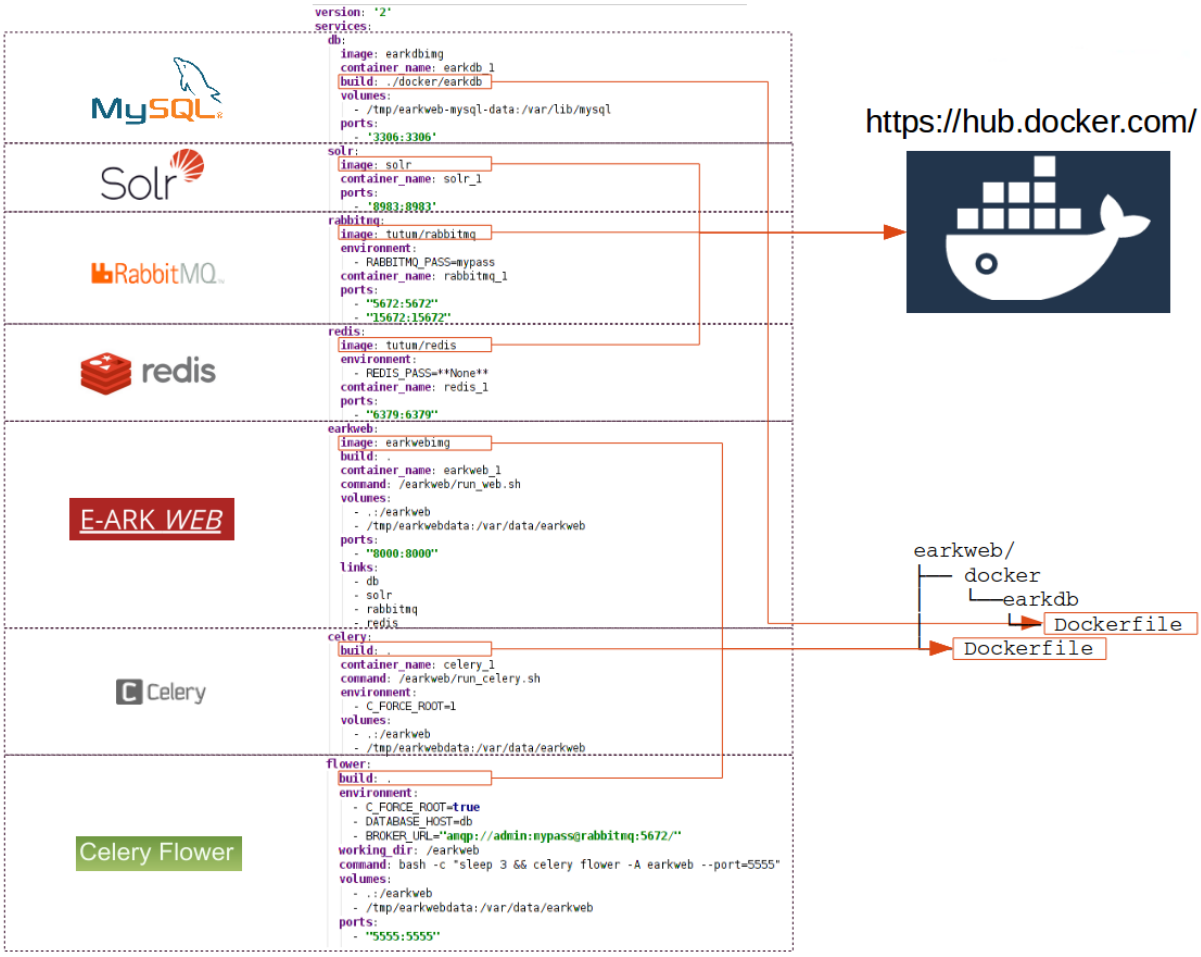


Figure 6 Origin of the images which are the basis to run the Docker containers

Finally, Figure 7 shows how various directories of the host file system are mapped to directories of the corresponding Docker containers. On the top right, there are directories of the local file system, and on the bottom right, there are directories of the Docker containers. The colours of the bounding boxes show the relationship between host file system and container directories. For example, the '/tmp/earkweb-mysql-data' directory of the host file system is mapped to the '/var/lib/mysql' directory of the MySQL database container. It is worth highlighting that for the containers *earkweb*, *Celery*, and *Celery Flower* the current directory ('.') –

[33] https://docs.docker.com/engine/reference/builder
[34] http://github.com/eark-project/earkweb

which corresponds to the *earkweb* code base – is mounted to the /earkweb container directory. These containers provide the different types of services (the *earkweb* frontend, the Celery task queue backend, and the Celery Flower task execution monitoring service) based on the same base image using service specific container run instructions respectively.

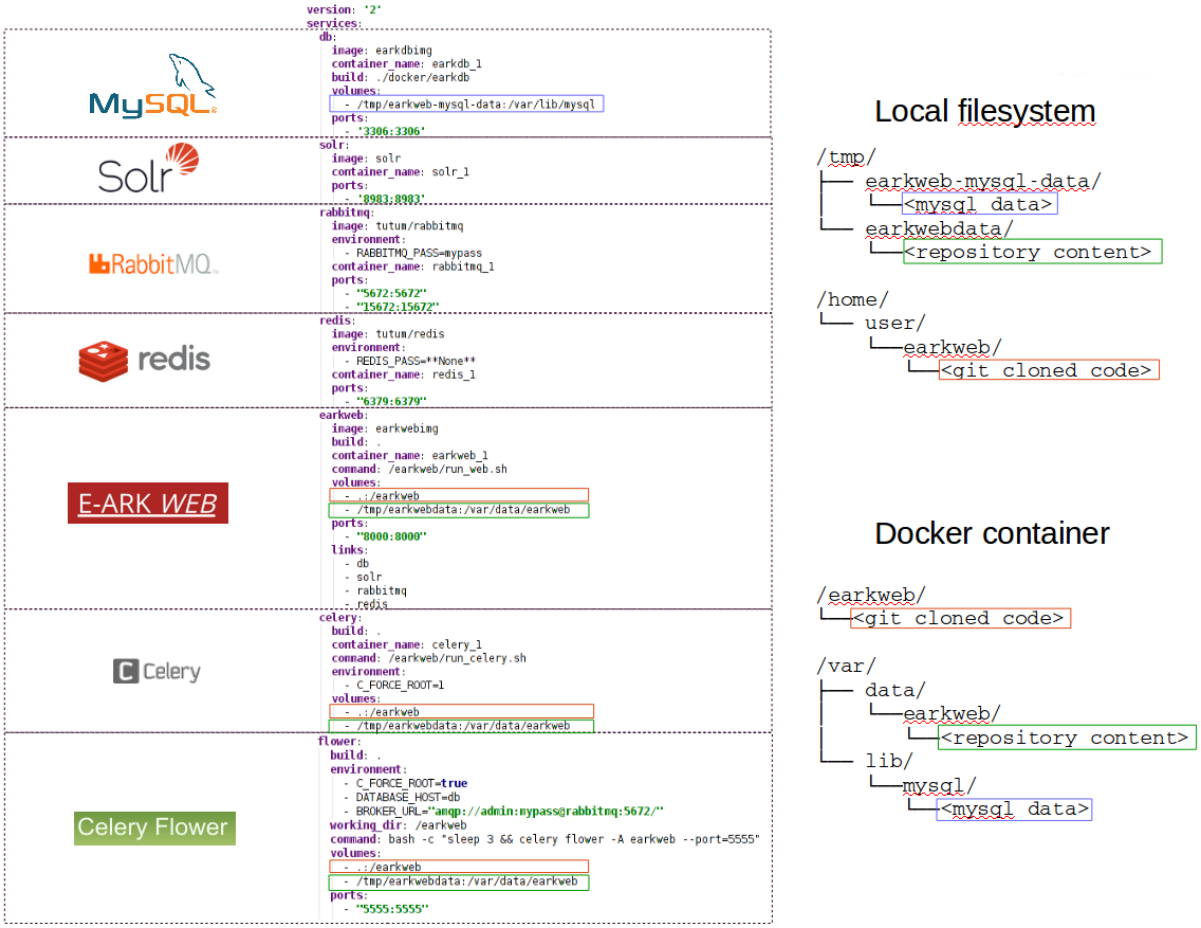


Figure 7 Data directories of the Docker containers and mapping to directories of the host file system

5 SIP-AIP conversion component

5.1 Task execution framework

The SIP-AIP conversion component consists of a set of individual tasks which are executed in a specific order to convert an E-ARK submission information package (SIP) into the E-ARK archival information package (AIP). It is an extensible workflow which can be adapted to specific needs by inserting new tasks at any point of the

workflow. *earkweb* uses a modular approach for defining atomic tasks which perform a specific transformation step of the SIP-AIP conversion, such as the extraction of an SIP or the validation of the descriptive metadata it contains. However, a specific task does not necessarily execute one single action, but can initiate a series of tasks or a complete workflow as well.

Each tasks is implemented as a python class and is available in the python module "workers/tasks.py"[35] of the *earkweb* application. A task which performs a step of the SIP-AIP conversion must extend the default task class DefaultTask defined in the module "workers/default_task.py".[36]

The default task makes sure that the pre-conditions for executing a task are fulfilled (e.g. the package is not in an error state), and also records the task execution in the provenance metadata of the package (i.e. the PREMIS metadata file). The default task also verifies if task execution is allowed given the current state of the package. Each task has a property which defines the list of tasks which are accepted as previously executed tasks. The fact that a task is defined as an "accepted last task" means that if execution was successful, there is the assumption that it produces valid output to be used as input for the current task. For example, to execute the SIPValidation task[37] which validates the structure of the E-ARK IP, it is a requirement that the transferred entity (the SIP's TAR file) was extracted successfully by the SIPExtraction task.[38]

5.2 SIP-AIP conversion tasks

Table 2 provides an overview of the tasks which together represent the SIP-AIP conversion component.

Task name	Accepted inputs	Task description
SIPDeliveryValidation	SIPValidation, SIPtoAIPReset	Validation of the delivery of the ZIP or TAR file against a delivery METS XML file to verify the integrity of the transferred file.
IdentifierAssignment	SIPDeliveryValidation	Assignment of a unique identifier. In the reference

[35] https://github.com/eark-project/earkweb/blob/master/workers/tasks.py
[36] https://github.com/eark-project/earkweb/blob/master/workers/default_task.py
[37] https://github.com/eark-project/earkweb/blob/6be44bc8ed3d346141c7f7e4091f1069c7a467f5/workers/tasks.py#L1100
[38] https://github.com/eark-project/earkweb/blob/6be44bc8ed3d346141c7f7e4091f1069c7a467f5/workers/tasks.py#L1059

		implementation a URN with the namespace identifier "uuid" is generated.
SIPExtraction	IdentifierAssignment	Extraction of the TAR or ZIP file containing the SIP.
SIPValidation	SIPExtraction	Validation of the structure of the SIP as well as the structural (METS) and preservation (PREMIS) metadata.
SIPRestructuring	SIPValidation, SIPExtraction	Restructuring of the SIP according to the AIP structure which encloses the SIP.
AIPDescriptiveMetadataValidation	AIPDescriptiveMetadataValidation, SIPRestructuring	Validation of descriptive metadata. In the current state, only EAD descriptive metadata is validated.
AIPMigrations	SIPRestructuring, AIPDescriptiveMetadataValidation, MigrationProcess, AIPMigrations, AIPCheckMigrationProgress	Execute AIP migrations according to defined migration rules. As examples, PDF to PDF/A and GIF to TIFF migrations are defined with corresponding tools performing the migrations.
AIPCheckMigrationProgress	AIPCheckMigrationProgress, AIPMigrations, MigrationProcess	This task can be invoked to query the migration process.
CreatePremisAfterMigration	CreatePremisAfterMigration, AIPCheckMigrationProgress	Create PREMIS metadata after migration.
AIPRepresentationMetsCreation	AIPRepresentationMetsCreation, CreatePremisAfterMigration	Create the METS structural metadata file for each representation contained in the AIP.

AIPPackageMetsCreation	AIPPackageMetsCreation, AIPRepresentationMetsCreation, CreatePremisAfterMigration	Create the AIP METS file referencing the METS files of the representations contained in the AIP.
AIPValidation	AIPValidation, AIPPackageMetsCreation	Validation of the structure of the AIP as well as the structural (METS) and preservation (PREMIS) metadata.
AIPPackaging	AIPPackaging, AIPValidation	Packaging the AIP as a TAR file.
AIPStore	AIPStore, AIPPackaging	Store the AIP in the file system in the Pairtree storage. This is the storage area of the standalone software stack. In the cluster software stack this storage area represents the staging area holding packages which are to be uploaded to the Lily Repository.[39]
LilyHDFSUpload	AIPStore	Upload AIP package file to the Lily Repository – only used in the cluster deployment stack.

Table 2 SIP-AIP conversion tasks

[39] The files are uploaded to the Hadoop Distributed File System (HDFS) of the Lily Repository. See deliverable D6.2 for details about the Lily Repository at http://eark-project.com/resources/project-deliverables/54-d62intplatformref-1

6 E-ARK Web user guide

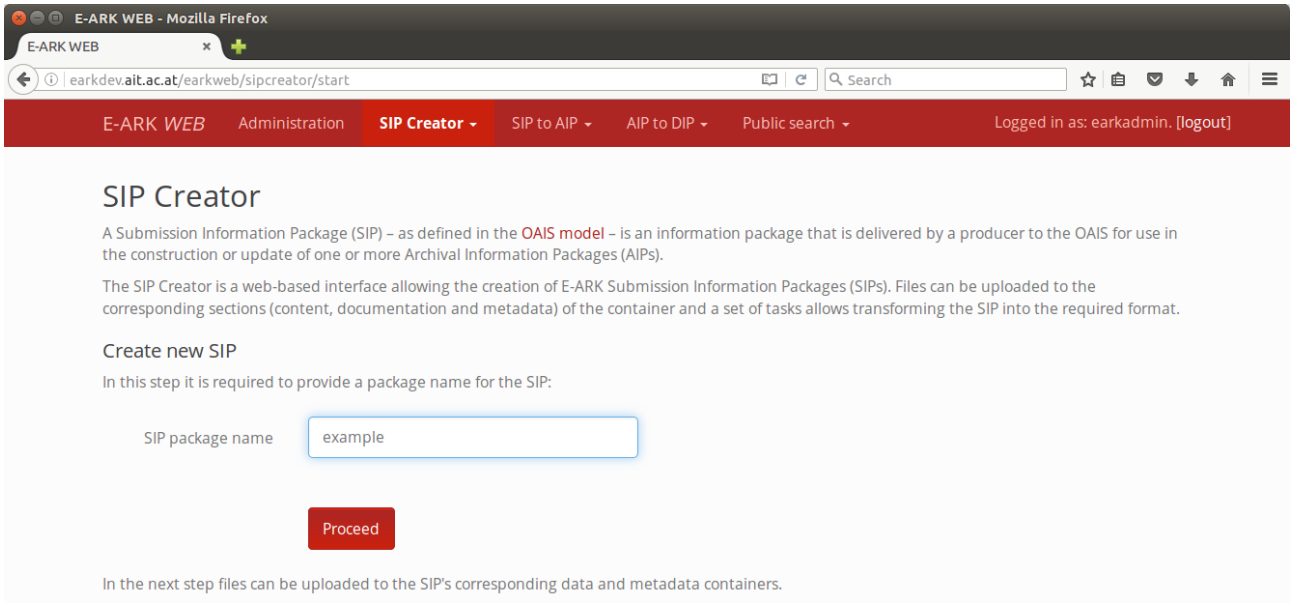
6.1 SIP Creator

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.

6.1.1 Initialise SIP package creation

In this step it is required that a package name for the SIP be provided:



Note that a package name must have at least 3 characters.

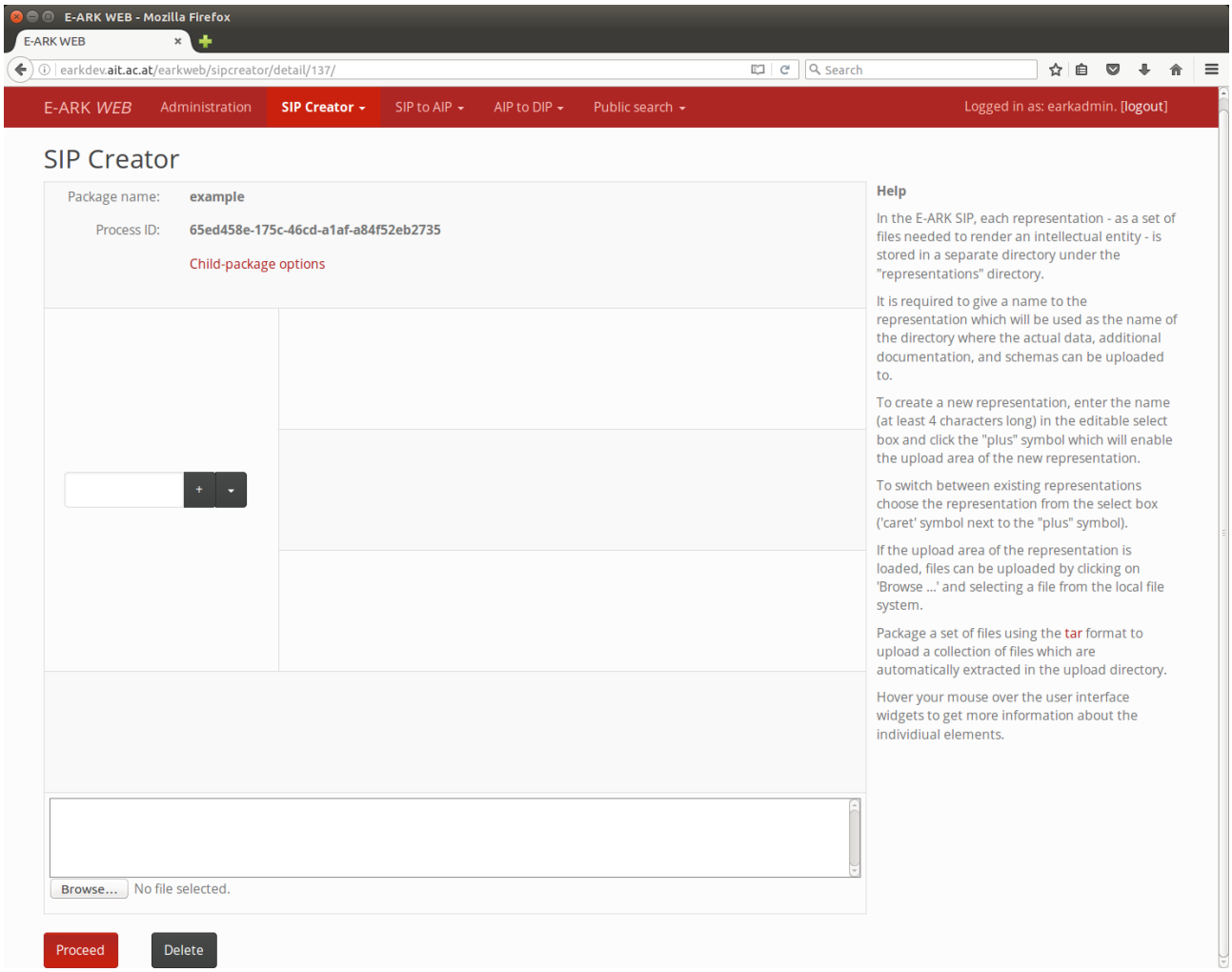
Click on 'Proceed' to continue with the next step.

6.1.2 Adding files to the SIP

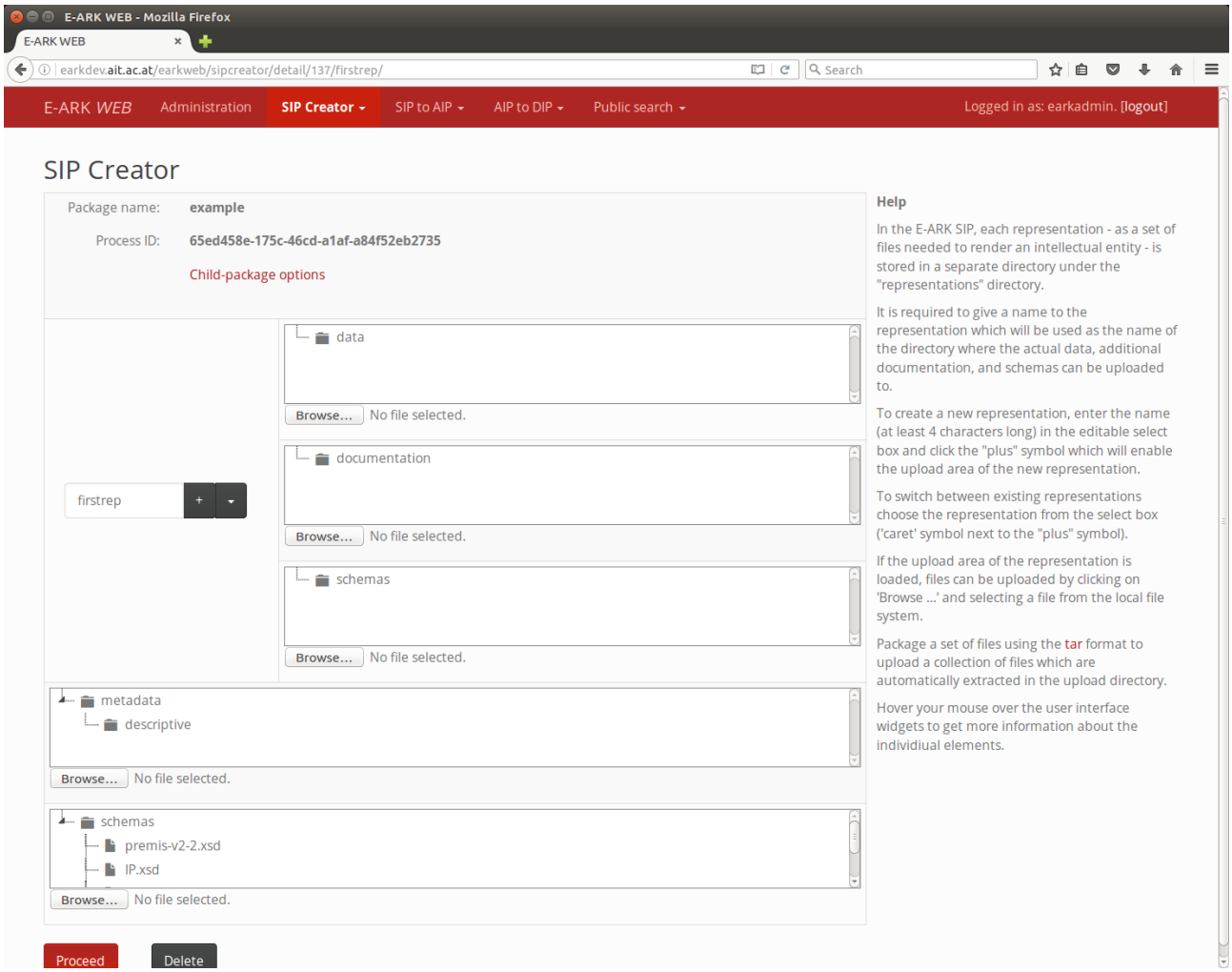
A web form allows uploading files to the corresponding areas of the SIP.

In the E-ARK SIP, each representation - as a set of files needed to render an intellectual entity - is stored in a separate directory under the "representations" directory.

It is required that a name be given to the representation which will be used as the name of the directory where the actual data, additional documentation, and schemas can be uploaded.



To create a new representation, enter the name (at least 4 characters long) in the editable select box and click the "plus" symbol which will enable the upload area of the new representation:



To switch between existing representations choose the representation from the select box ('caret' symbol next to the "plus" symbol).

If the upload area of the representation is loaded, files can be uploaded by clicking on 'Browse ...' and selecting a file from the local file system.

Package a set of files using the tar format to upload a collection of files which are automatically extracted in the upload directory.

Hover your mouse over the user interface widgets to get more information about the individual elements.

Click on the "Proceed" button to continue with the next step.

6.1.3 SIP creation process

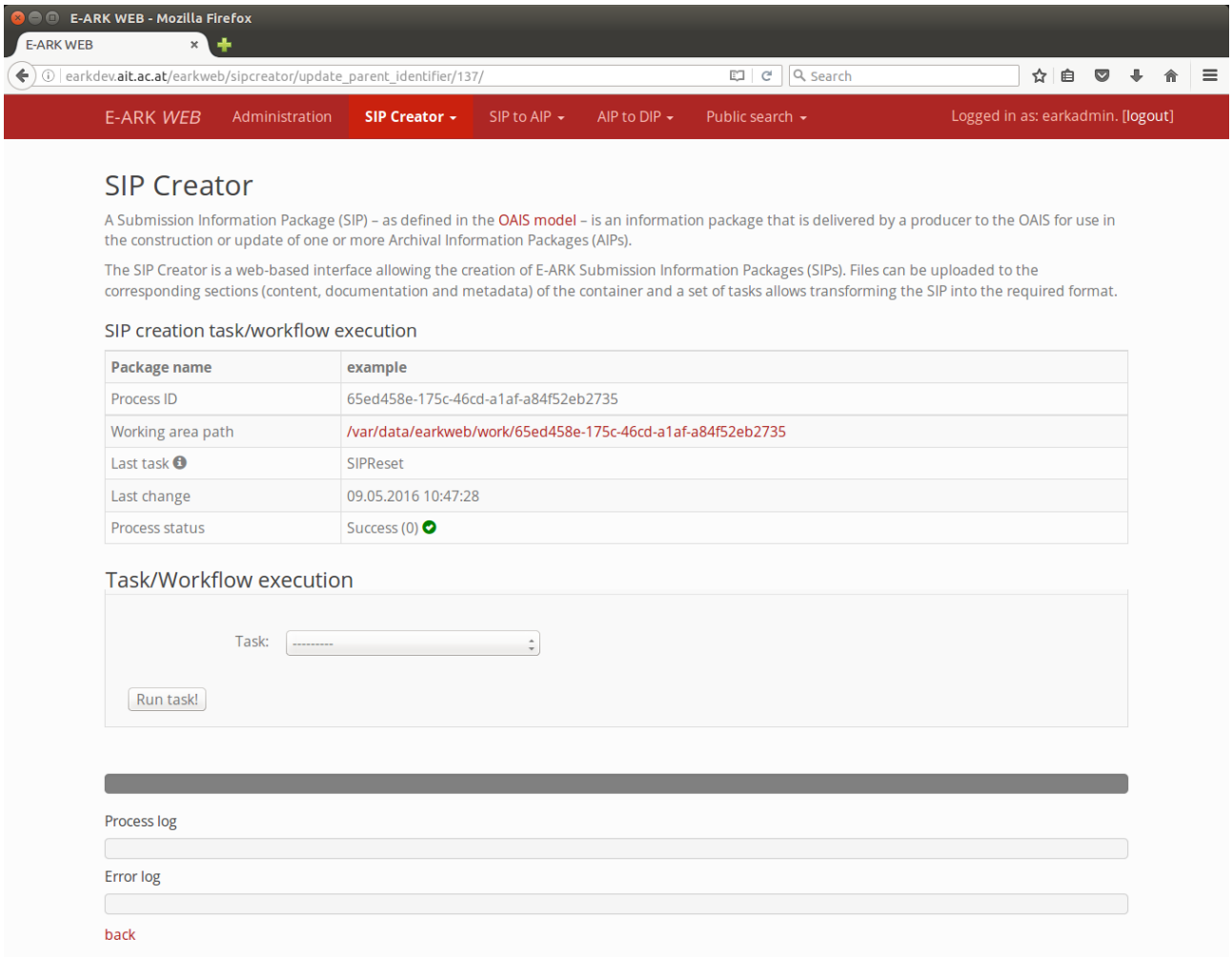
6.1.3.1 Start SIP creation process

The SIP creation process interface allows executing SIP creation tasks:

- SIPReset
- SIPDescriptiveMetadataValidation
- SIPPackageMetadataCreation
- SIPPackaging
- SIPClose

6.1.3.1.1 Task execution

The tasks in the pull-down menu near the label "Tasks" appear in the logical order in which they should be executed.



The "SIP creation task/workflow execution" overview table shows the information entities of the package:

- The "Package name" is the name of the SIP which is the outcome of the current SIP creation process.
- The "Process ID" is a UUID which corresponds to the name of the working area directory.
- The "Working area path" is a link which allows accessing the working area directory of the corresponding package to see the result of a task execution.
- The "Last task" shows the last task which was executed, e.g. "SIPValidation". The last task also determines the following task which can be executed because each task defines a list of accepted input tasks.
- Last change is the date/time of the last modification (by a task)
- Process status shows the consistency status of a package. If the process status shows the value

"Success (0)" together with a green "check mark" symbol, then the status of the package is consistent. If an error occurred during processing, the process status shows the value "Error (1)" together with a green "warning triangle" symbol.

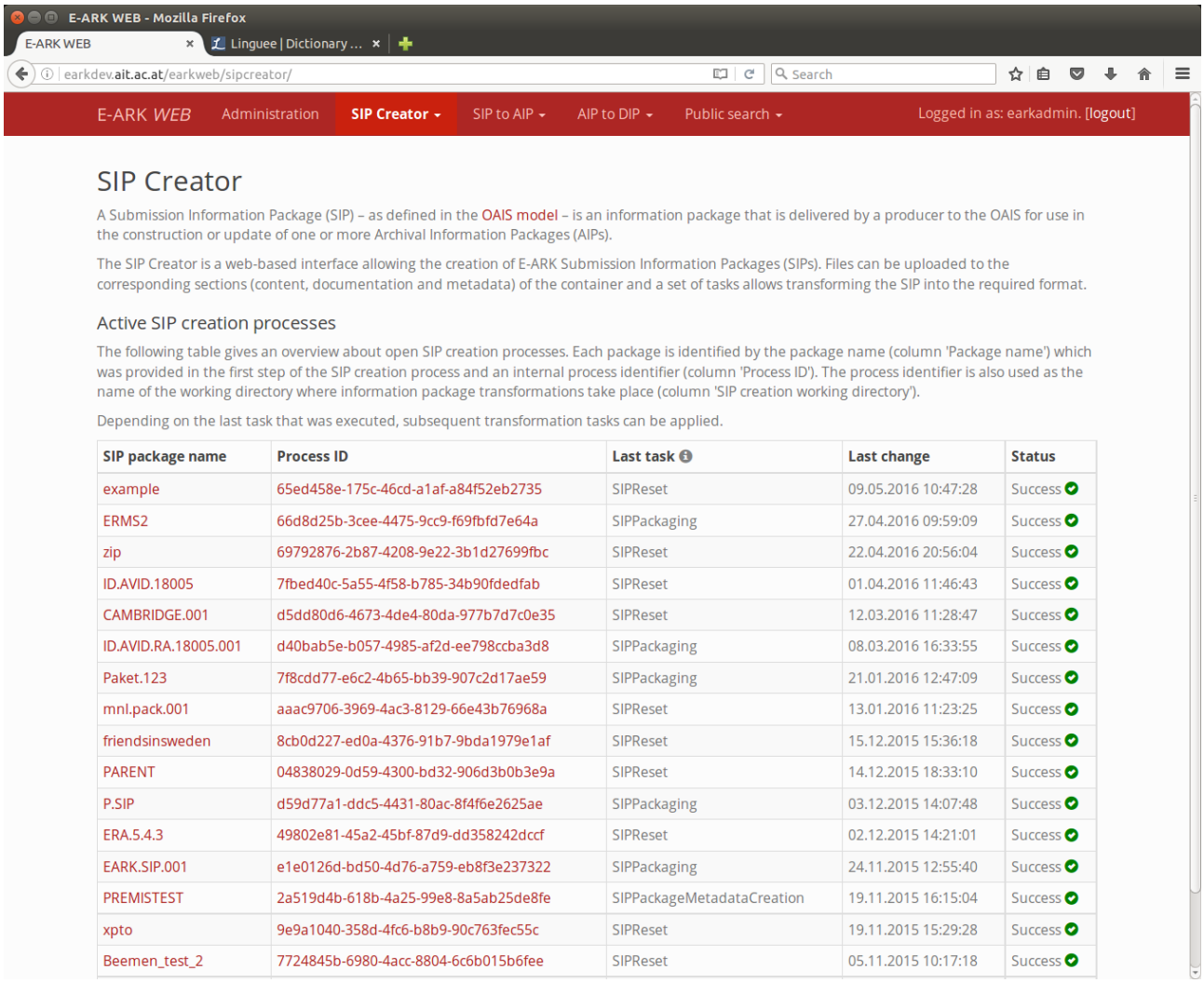
Selected information and error log messages appear in the "Process log" and "Error log" areas, more detailed information about the processing might be available in the *earkweb* processing log which is available in the package.

Use the "SIPReset" task to roll-back package processing to the initial state in case an error occurred during processing (after applying required modifications).

It is possible to directly continue with the "SIP to AIP conversion" process by executing the "SIP Close" task. Note that in this case the package disappears from the "Active SIP creation processes" overview page as it is handed over to the "SIP to AIP conversion".

6.1.3.2 Active SIP creation processes overview

By accessing the sub-menu item "Active SIP creation processes", an overview about open SIP creation processes is shown. Each package is identified by the package name (column 'Package name') which was provided in the first step of the SIP creation process and an internal process identifier (column 'Process ID'). The process identifier is also used as the name of the working directory where information package transformations take place (column 'SIP creation working directory').



Depending on the last task that was executed, subsequent transformation tasks can be applied.

6.2 SIP to AIP conversion

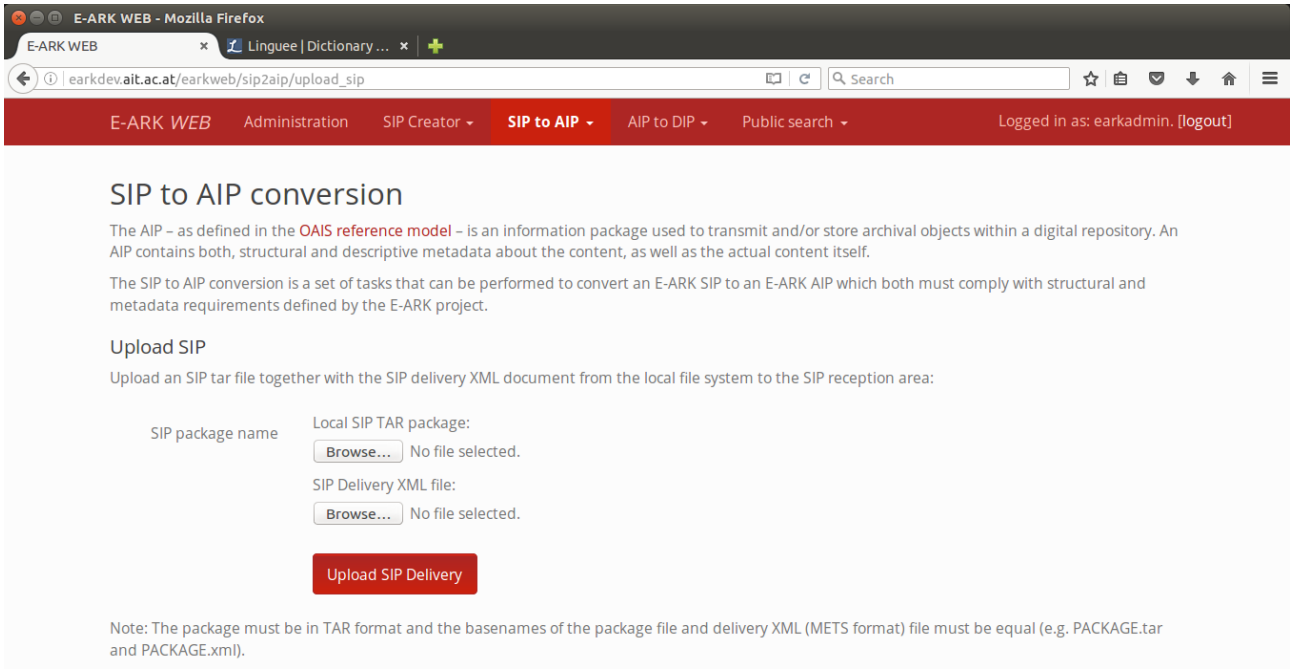
The AIP – as defined in the OAIS reference model – is an information package used to transmit and/or store archival objects within a digital repository. An AIP contains both, structural and descriptive metadata about the content, as well as the actual content itself.

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 must comply with both structural and metadata requirements defined by the E-ARK project.

6.2.1 Start SIP to AIP conversion process

The "SIP to AIP conversion" is either started by handing the package over from the "SIP creation" process

within *earkweb* by executing the "SIP Close" task or it can be uploaded using the "SIP to AIP conversion" sub-menu item "Upload SIP":



The TAR or ZIP container file of an SIP is uploaded together with the SIP delivery XML document from the local file system to the SIP reception area.

Note that the package must be in TAR or ZIP format and the basenames of the package file and delivery XML (METS format) file must be equal (e.g. PACKAGE.tar and PACKAGE.xml).

6.2.1.1 SIP to AIP conversion tasks

The SIP to AIP conversion process interface allows executing SIP creation tasks:

- SIPtoAIPReset - Rollback to initial state (as after handover from SIP creation or uploading the SIP)
- SIPDeliveryValidation - Validate delivery (especially checksum information)
- IdentifierAssignment - Assign identifier
- SIPExtraction - Extract SIP
- SIPRestructuring - Restructure SIP according to AIP format
- SIPValidation - Validate SIP
- AIPMigrations - Do migrations according to predefined rules (e.g. PDF to PDF/A)

- AIPCheckMigrationProgress - Check if migrations are finished
- CreatePremisAfterMigration - Create PREMIS file after migrations completed
- AIPRepresentationMetsCreation - Create representation METS files
- AIPPackageMetsCreation - Create AIP METS and PREMIS
- AIPValidation - Validate package
- AIPPackaging - Create package container file
- AIPStore - Store AIP package in file system storage area (pairtree storage)
- AIPIndexing - Index AIP
- LilyHDFSUpload - Upload AIP to public search server

6.2.1.2 SIP to AIP conversion task execution

The tasks in the pull-down menu near the label "Tasks" appear in the logical order in which they should be executed.

The "SIP to AIP task/workflow execution" overview table shows the information entities of the package. Note that some information entities, such as "identifier" might appear only after a specific task ("IdentifierAssignment" in this case), was executed.

- The "Process ID" is a UUID which corresponds to the name of the working area directory.
- The "Package name" is the name of the SIP which is the outcome of the current SIP creation process.
- The package Identifier is the PUID of the AIP. Although of type UUID in the reference implementation, this could be any other type of identifier, such as a DOI or PURL.
- The "Working area path" is a link which allows accessing the working area directory of the corresponding package to see the result of a task execution.
- The "Last task" shows the last task which was executed, e.g. "AIPValidation". The last task also determines the subsequent task which can be executed because each task defines a list of accepted input tasks.
- Last change is the date/time of the last modification (by a task).
- Process status shows the consistency status of a package. If the process status shows the value "Success (0)" together with a green "check mark" symbol, then the status of the package is consistent.

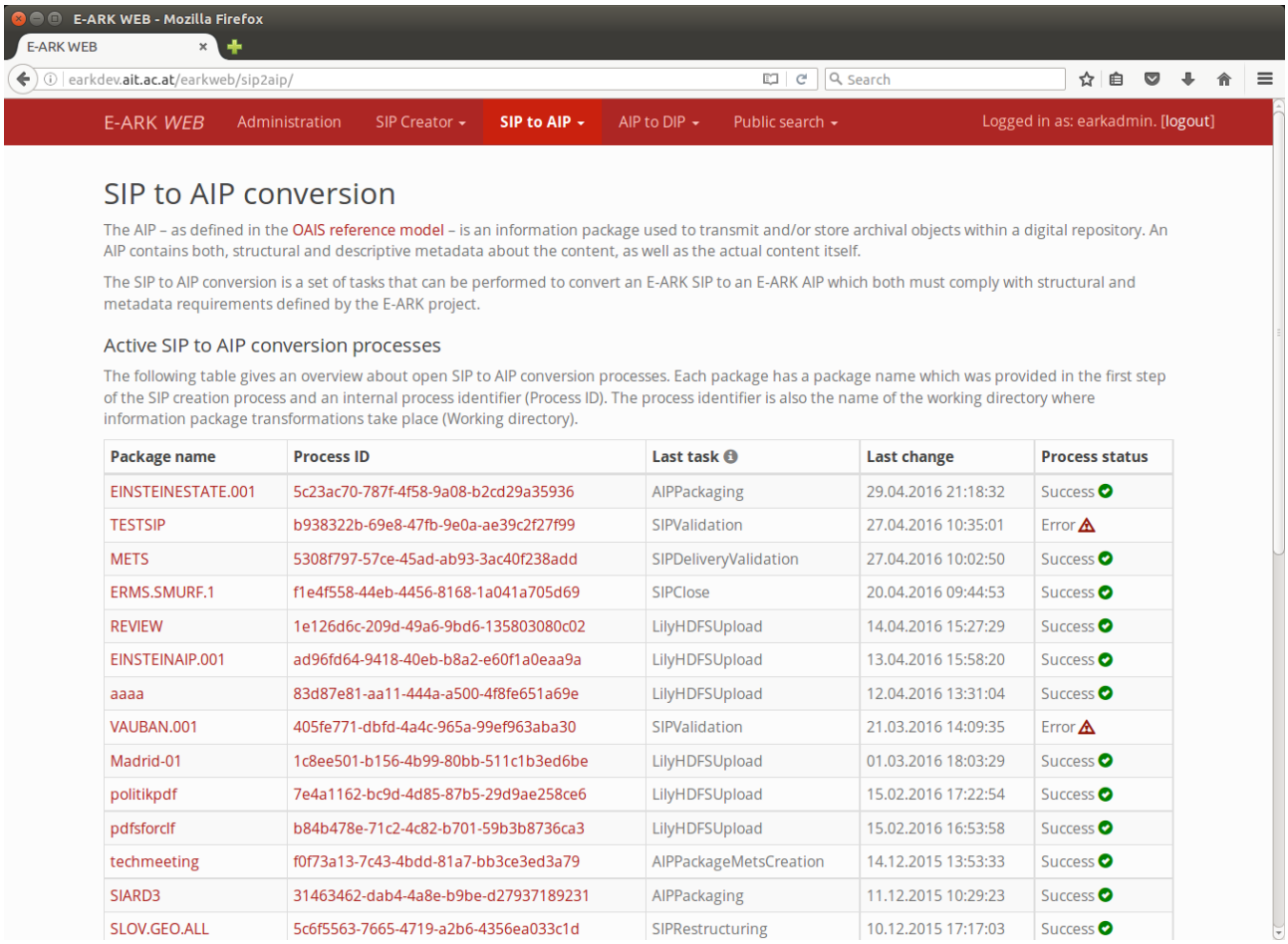
If an error occurred during processing, the process status shows the value "Error (1)" together with a green "warning triangle" symbol.

Selected information and error log messages appear in the "Process log" and "Error log" areas, more detailed information about the processing might be available in the *earkweb* processing log which is available in the package.

Use the "SIPtoAIPReset" task to roll-back package processing to the initial state in case an error occurred during processing (after applying required modifications).

6.2.1.3 Active SIP to AIP conversion processes

By accessing the sub-menu item "Active SIP to AIP conversion processes", an overview about open SIP to AIP conversion processes is shown. Each package is identified by the package name (column 'Package name') which corresponds to the name of the SIP container. The process identifier is also used as the name of the working directory where information package transformations take place (column 'SIP creation working directory').



Depending on the last task that was executed, subsequent transformation tasks can be applied.

6.2.1.4 Indexing and search in AIPs

The "AIPIndexing" task allows indexing the AIP TAR package created the SIP to AIP conversion process. The AIP is indexed according to the location in the pairtree storage. In that way the search results can offer a link to directly render individual content items.

6.2.1.5 AIPs in pairtree storage

The storage backend used in the Standalone Software Stack[40] of *earkweb* is based on the Python-based *Pairtree File System* implementation[41] of the *Pairtrees for Collection Storage* specification[42]. It is used to store the physical representation of the AIP in a conventional file system as opposed to the Hadoop Distributed File System which is used in the Revised Cluster Software Stack described in Deliverable D6.3.

In summary, the *Pairtree* is a filesystem hierarchy for storing digital objects where the unique identifier string of the object is mapped to a unique directory path so that the file system location of the object can be derived from the identifier string. Basically, the identifier string is split each two characters at a time and the object folder has by definition more than two characters. Furthermore, the specification defines a mapping of special characters to a set of alternative characters in order to ensure file system level interoperability.

The following example will explain how the *Pairtree* storage method is used in the *earkweb* implementation.

According to the default *earkweb* configuration, the path to the storage directory in the file system is:

```
/var/data/earkweb/storage
```

The storage folder contains an empty file called "`pairtree_version0_1`" which specifies the version of the *Pairtree* specification.

The storage directory contains the root folder of the *Pairtree* storage:

```
/var/data/earkweb/storage/pairtree_root
```

Let us assume that an AIP has the following identifier

```
urn:uuid:6c496473-4e77-44f5-b387-25bffd362789
```

Following the method defined by the *Pairtree* specification, this identifier is mapped to the following path:

```
ur/n+/uu/id/+6/c4/96/47/3-/4e/77/-4/4f/5-/b3/87/-2/5b/ff/d3/62/78/9
```

and the actual AIP container file is stored in a "data" folder which represents the leaf node of the *Pairtree* file system hierarchy.

The leaf node contains one or possibly more sub-directories (5-digits fixed length zero-filled number) for the versions of the AIP.

[40] In contrast to the cluster software stack, the standalone software stack does not include a big data backend based on Apache Hadoop.
[41] https://pypi.python.org/pypi/Pairtree
[42] https://wiki.ucop.edu/display/Curation/PairTree?preview=/14254128/16973838/PairtreeSpec.pdf

The full path to the AIP file is as follows (to be read as a single line string):

```
/var/data/earkweb/storage/pairtree_root/ur/n+/uu/id/+6/c4/96/47/3-/4e/77/-
4/4f/5-/b3/87/-2/5b/ff/d3/62/78/9/data/00001/urn+uuid
+6c496473-4e77-44f5-b387-25bffd362789.tar
```

Note that the colon characters (":") which are part of the URN identifier to separate the leading "urn" sequence from the namespace identifier ("uuid") and the latter from the namespace-specific string ("6c496473-4e77-44f5-b387-25bffd362789") were mapped to the plus character ("+"). This is because the *Pairtree* character mapping rules apply to the filename of the physical container as well to ensure file system interoperability.

The purpose of the *Pairtree* storage backend is to allow a large number of AIPs[43] to be stored in a conventional file system, such as Network Attached Storage, for example, and allow fast access to individual files by directly reading content streams from the files stored in the TAR packaged AIP container files.

If the AIP is changed, because, for example, a new representation was added to the AIP, a new physical container will be created as a new version of the AIP. As a result, only the numerical folder will be incremented (00002) and the identifier for the intellectual entity remains the same (`426087e8-0f79-11e3-847a-34e6d700c47b`):

```
/var/data/earkweb/storage/pairtree_root/ur/n+/uu/id/+6/c4/96/47/3-/4e/77/-
4/4f/5-/b3/87/-2/5b/ff/d3/62/78/9/data/00002/urn+uuid
+6c496473-4e77-44f5-b387-25bffd362789.tar
```

6.2.1.6 Parent-child AIP relations

During AIP creation, it is possible to assign parent-child relationships between AIPs, as described in D4.4 part A. Figure 8 illustrates how the creation of packages governed by a parent-child relationship can be achieved:

1. The submission of a package is processed up to the point where the identifier of the parent AIP ("parentID") is created (operation "submitParentSIP").
2. This SIP to AIP creation process remains open until the last child AIP is created.

[43] No concrete numbers will be given here about what a "large number of AIPs" means in this context, as the requirements regarding scalability of indexing and access procedures depend on various factors, such as the environment (hardware and network) and the type of collection data to be stored in the repository. It is a matter of evaluating the Standalone Software Stack (cf. section **Error! Reference source not found.**) using scalability tests to find out if it can cope with the given scalability requirements, or if the Cluster Software Stack (cf. section **Error! Reference source not found.**) is needed.

3. A sequence of child SIPs can then be submitted and created as child-AIPs by indicating the parent identifier ("parentID") generated previously.
4. After the last child-AIP is processed, the parent-AIP is created (operation "createParentAIP") by indicating which child-AIPs belong to the AIP ("childs").
5. Finally, the consistency of the logical AIP composition is verified and the parent-AIP ("parent") and all child-AIPs ("childs") are stored (operation "storeAIP").

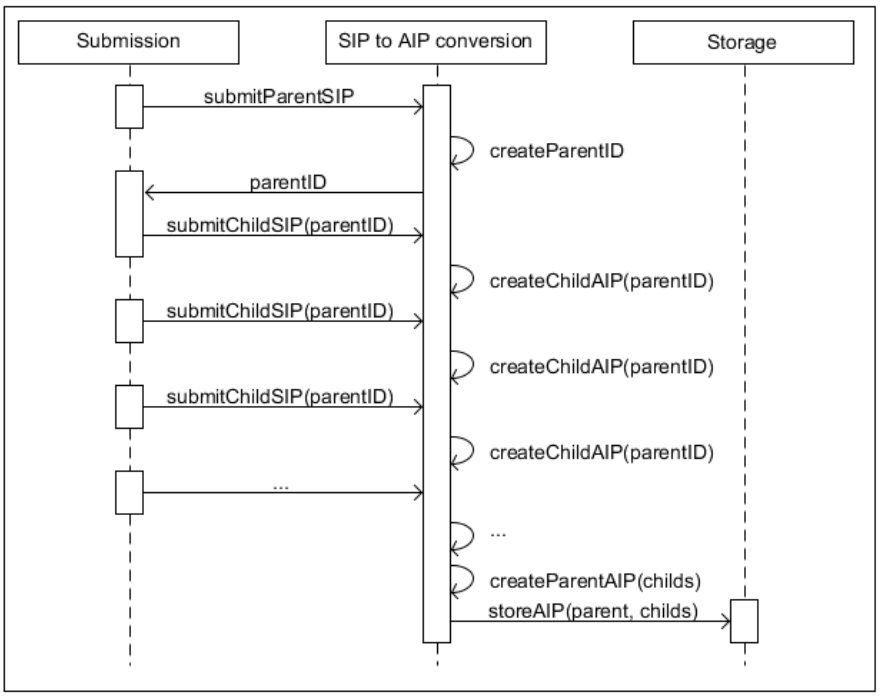


Figure 8: Parent-child aggregation submission process