D.4.2 Archival Information Package (AIP) Draft Specification

DOI: 10.5281/zenodo.1172685

Grant Agreement Number:	620998
Project Title:	European Archival Records and Knowledge Preservation
Release Date:	13th February 2018

Contributors

Name	Affiliation
Jan Rörden	University of Cologne
Manfred Thaller	University of Cologne
Torben Lauritzen	Magenta ApS
Kuldar Aas	National Archives of Estonia
Janet Anderson	University of Brighton
David 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This deliverable contains original unpublished work except where clearly indicated otherwise.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,

TABLE OF CONTENTS

–

List of figures

Executive Summary

This document builds on the overview of existing solutions for AIPs given in D4.1. It describes a blueprint for the structure of an AIP format following from the state of the art described there.

As an AIP has a potentially unlimited life span, however, we augment the presentation of the format of the physical file representing the AIP by a discussion of how such an AIP may keep an unchanged identity, while its physical representation may change over time.

A “Pan-European AIP format” is supposed to handle essentially each type of digital content a user may want it to contain. It is obviously impossible, to describe in a format document, how content not yet known will be handled. The document, therefore, also contains a chapter on the way in which this format is embedded into the technical workflow within which an AIP exists.

1. Introduction

The present work – D4.2: The E-ARK AIP draft specification – is part of the Task *4.1 : E-ARK-AIP specification within Work Package 4: Archival Records Preservation of the E-ARK Project*. It continues the considerations begun in deliverable *D4.1: Report on available formats and restrictions* and is closely connected to it. While D4.1 described existing AIP formats, this deliverable now proposes a structure, which creates a first draft of a new AIP format, and which shall confer the advantages of the formats described in the previous deliverable and fulfil the criteria for a Pan-European AIP format identified there.

AIP – *Archival Information Package* – is used throughout this document as defined in the Reference Model for an Open Archival Information System OAIS, CCSDS 650.0-M-2, 2012 ("Magenta Book"), pp. 2-7 / 2-8.

An "AIP format" describes first of all a format in the usual sense, i.e., a set of rules, showing how information is to be stored on some media and to be accessed by appropriate software. We describe this view of the AIP format in chapter "2. Static View of an AIP". In this view the format of an AIP is very close to the format of a file. An AIP, however, also represents an entity, which may be augmented over time, without losing its identity. Considerations of this "identity" are not usually part of a file format specification. These provisions for these identity related questions are described in the chapter "3. Dynamic View of an AIP".

Each individual implementation of the AIP might be slightly different depending on technical preferences and decisions at individual institutions. Therefore chapter 4 will discuss how these decisions might affect the AIP format and the SIP2AIP conversion tool needed to support the format and ultimately present the outline of a generic API of a SIP2AIP tool. As this is a draft – the first of three iterations towards the Pan-European AIP format – the style of this document is not exclusively declarative, as a format specification in the strict sense should be, but is combined with explicit reasoning about the details of the format proposed here.

This document describes a concrete format, to be supported and implemented by software developed within E-ARK. As with other format specifications, it does not usually refer to other documents or specifications, but rather concentrates on the decisions proposed for implementation. To connect our decision on technology and design, we would like to refer to D4.1, chapter: "A Pan-European AIP format: The vision.", pages 6-10. This document can be seen as a description of how we propose to act on the effects of the "vision" for the E-ARK AIP design described there.

A short note on two concepts, which may be confusing: The terms "version" and "generation" have slightly different implications in chapters 2.1.1. and 2.1.2, which can only be discussed if the reader has some knowledge about these chapters. They are discussed in chapter 2.1.3, therefore. It may be useful, to first read these chapters in succession, and then re-read 2.1.1 and 2.1.2 after having read 2.1.3.

2. Static View of an AIP

2.1. Abstract Description

2.1.1. The General Case

In Deliverable D4.1 it has been required that an AIP should be physically, as well as logically, autonomous. *Physical autonomy* requires an AIP to be an integrated byte stream, which can be handled independently from any repository system and be processed by ubiquitously available standard tools. *Logical autonomy* is reached when an AIP contains all the metadata which are necessary to interpret the content of the AIP, even if no additional information about it is available outside the AIP [cf. D4.1: p.8-10].

On the physical level, that implies that an AIP is represented by a *robust container*, which is separated into different segments. A *robust container* is a continuous byte stream, which can be handled by widely available tools, independent of any proprietary software component of a repository. It contains the parts of an AIP integrated into such a bytestream in a widely supported format, which allows the extraction of parts of its content, even if the bytestream has become corrupted. For the time being uncompressed tarballs will be used. If tar[1] becomes obsolete, a repository is expected to repackage the content of the tar-balls in a new container format. This operation is not covered by this document. Within the robust container an AIP is a tree of folders and files, which is packaged as an uncompressed tar-ball [cf. D4.1: p. 37-38] for storage within a repository. This tree of folders has the abstract branches shown in Figure 1. We would like to emphasize, that we are describing a generic specification here, which is content agnostic. I.e., all definitions proposed in this chapter are supposed to cover a collection of images, as well as a single PDF file or a database etc. to be preserved.

We start by describing the components in the *full* structure presented in Figure 1. There are, however, considerable variations possible in implementing it. These variations are described in chapter 2.1.2 below. To keep the various possibilities of implementation clearly separated, we speak of a *compound* AIP when an implementation decides to store all the components discussed below within *one* physical container for an AIP.

[1] http://en.wikipedia.org/wiki/Tar_(computing)

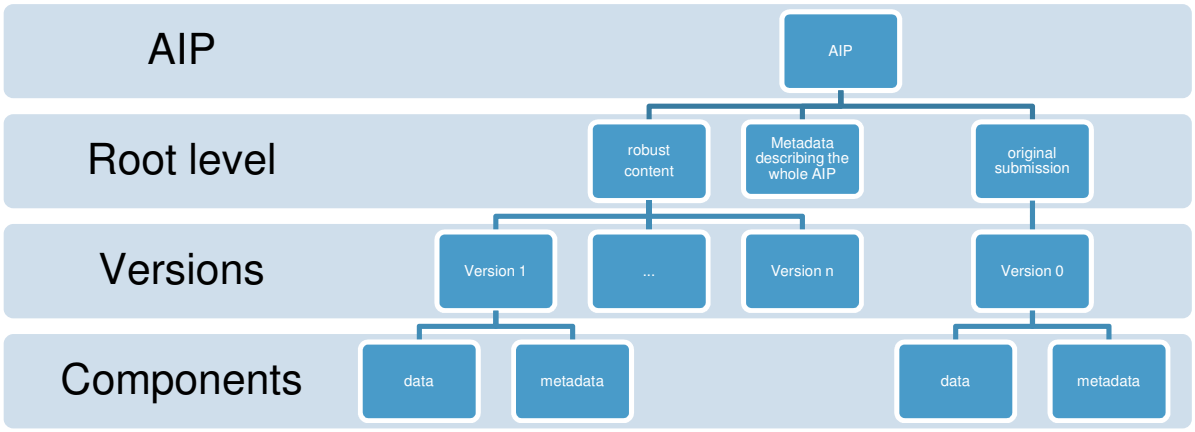

Figure 1: Abstract AIP structure

Within this tree at the root level of the container, three branches are distinguished. The *robust content* branch contains the data and metadata that make up the information to be preserved. All versions fork into *data* and *metadata* branches. To avoid an overly crowded diagram, we present them for versions 0 and 1 only. A first version of such a *robust content* will frequently be created during ingest. It contains data and metadata that guarantee:

1. Completeness: All data contained in a *robust content version* are described within the metadata contained in the same *robust content version*. All data items described in the metadata are either contained within the same *robust content version*, or they are AIP identifiers of other AIPs contained within the same preservation system. The repository administering the AIPs is responsible to detect and defuse any cyclic references that are created by mistake.

2. Fitness for preservation: All data and metadata within a *robust content version* are stored in non-proprietary data and metadata formats, which, according to current knowledge, are appropriate for preservation purposes.

This obviously represents the primary purpose of preservation, so why will the first version be created "frequently" - and not necessarily "always" - during the creation of the AIP?

According to the current state of the art, it is not possible to guarantee that any specific migration from one file format to another one will be free of errors. Therefore the original byte streams representing the information package submitted will always be contained with the AIP. To increase the security of this part of the AIP, it is contained in its own branch at the lowest level.

If this version 0 of the byte streams has been submitted in data and metadata formats which are considered fit for preservation at the time of ingest, there is no reason to migrate the data and metadata, and the *original submission* branch of the AIP will contain the only version within the AIP.

If the data and/or metadata formats contained within the SIP are not considered fit for preservation, at least one migration of the formats will be performed during ingest and stored in the *version 1* branch of the robust content branch.

The AIP format described here is a technical format: it does not make any assumptions on the legal and technical framework at the data provider's site. Thus – you might have situations where the original data has already been altered / migrated by the data provider, or the data are provided to the archives "as is" in its original format. Nevertheless, the "Version 0" is still to be seen as the "submitted original" -> the thing that the archives receive regardless of the situation.

If there are any reasons to perform migrations into additional formats at the time of ingest, these are stored in additional *version n* branches of the robust content. This may happen, e.g., if during ingest of a relational database, the database shall be stored for operational characteristics in a set of normalized tables, but at the same time also in de-normalized form. This may be advisable, in cases where the possibility to consult personal knowledge of the creators / users of a database will enhance the quality of de-normalization. Such creators / users of a database might not be available any more for consultation at a later stage – say after 100 years - for biological reasons. In this case the resulting AIP would contain { *version 0* : the binary, *version 1*: normalized preservation format, e.g. SIARD, *version 2*: de-normalized preservation format }.

If at a later stage, a repository is instructed by the technology watch of the digital archive to drop certain data or metadata formats, which are on the brink of obsolescence, the AIP will be opened and the required migration will be performed, additional *version n* branches being created. If by this operation an older *version* branch becomes obsolete or redundant, it may be excluded from re-packaging after the later migration. Keeping in mind that the success of an earlier migration cannot be guaranteed, any migration will use as a starting point the oldest version that still can be processed.

Technology watch refers here to the institutional and trans-institutional precautions of the institution responsible for a digital archive to detect changes in the general IT environment, which may change the fitness of the digital content for preservation. Such changes can, e.g., arise when a file format becomes obsolete and there are no tools available which are able to process it.

Within each *version*, there exists a *metadata* branch which contains such metadata as are needed to process the content of the *data* branch of that version.

Above these branches, forking at the root level of the AIP, there is also a *metadata* branch which contains metadata related to the AIP as a whole. These metadata consist of:

1) Exactly one PREMIS[2] file, describing the cumulative history of the AIP, starting with a description of the original ingest process. Any modification to an AIP – as in the case of a migration described above – or a result of one of the operations described below in chapter 2 – leads to an update of the PREMIS file.

[2] http://www.loc.gov/standards/premis/

2) A set of files with the hash values each of which contains "hashes" related to one of the versions, occurring in the O*riginal submission* branch or the *robust content* branch of the AIP. (For simplicity's sake, we speak here of "hashes". We have not yet decided whether this will indeed be hashes in the technical sense, checksums or other signatures derived from a file to guarantee its integrity. We explicitly recommend that provisions be made to include such digital signatures as may be required by authorities in the future to guarantee the legal validity of stored documents.

3) One file representing the *ID context* (specified below) of the AIP.

Note: *Metadata related to the AIP as a whole describe the IP as a whole, that is, not only the content, but also its history. Metadata describing the content directly are themselves stored in technical formats, which may change for the same reasons as data formats change. They are considered part of the versions, therefore.*

Note: *The specification above assumes, that only events, which change the content of an AIP, have to be documented with the AIP. This has the advantage, that changes to the AIP will be very rare. One has to point to a side effect, however: If a complete record of AIP access is required, this has to be provided by the logfiles of the repository, in which the AIP is stored. Still, this does not imply adding new components to the AIP model on Figure 1 but rather extending and specifying additional metadata elements to strictly serve this purpose.*

Note: *At the current stage of planning, this specification covers only AIPs, which explicitly support preservation strategies based on migration. We strongly suggest for the next iterations of this specification to examine the possibility to store additional information in order to support emulation as a preservation strategy. This means that we would need to include a description of the tools used to create and access the content of the SIP. This includes, but is not restricted to, the hardware specification of the workstations (including peripherals), OS version and the software used (office programs etc.).*

The "ID context" of an AIP implements a concept to handle the relationship between AIPs, which is required to handle the notion of an AIC (*Archival Information Collection*). The importance of the "identity" of an AIP becomes clear only when discussed within the life cycle of an AIP. We leave, therefore, the discussion of this concept for chapter 2 below.

Statically the ID context of an AIP is represented by a file, which contains a complete set of the other AIPs, which have to be accessible to process an AIP that represents an AIC. This is best explained by an example.

Several of the archival partners of E-ARK represent content as a set of interconnected IPs. The most intuitive example is a finding aid, which represents a level in the general archival tectonic of an institution. The AIP representing this finding aid will have to refer to the AIPs described in the finding aid. The ID context of the AIP of the finding aid consists of a list of the AIPs referenced within this AIP, together with a flag indicating whether these AIPs have already been ingested into the repository holding the AIP of the finding aid (or: index) at the time of its own ingest.

As will become clear in the discussion of the dynamic view of an AIP, there are considerable differences in the processing of an AIP which can be processed without assumptions about the existence of other ones, and AIPs which do rely on the existence of other ones. As we can easily envisage cases where an AIP is not a "collection" in any literal sense, but relates to another AIP nevertheless, we do not consider the concept of an AIC as very central, but rather focus on the distinction between AIPs which do and AIPs which do not reference other AIPs. The AIC is then simply a special case of the former.

2.1.2. Variations of the General Case in Application

As discussed above, an AIP is called a compound AIP, if the complete structure described within 2.1.1 is stored within one physical container. For easier reference, Figure 2 below repeats Figure 1. In such an implementation there is usually one AIP representing one SIP. It contains the content of the original submission (SIP), migrated data/metadata (*robust content*) as well as metadata describing the whole AIP.

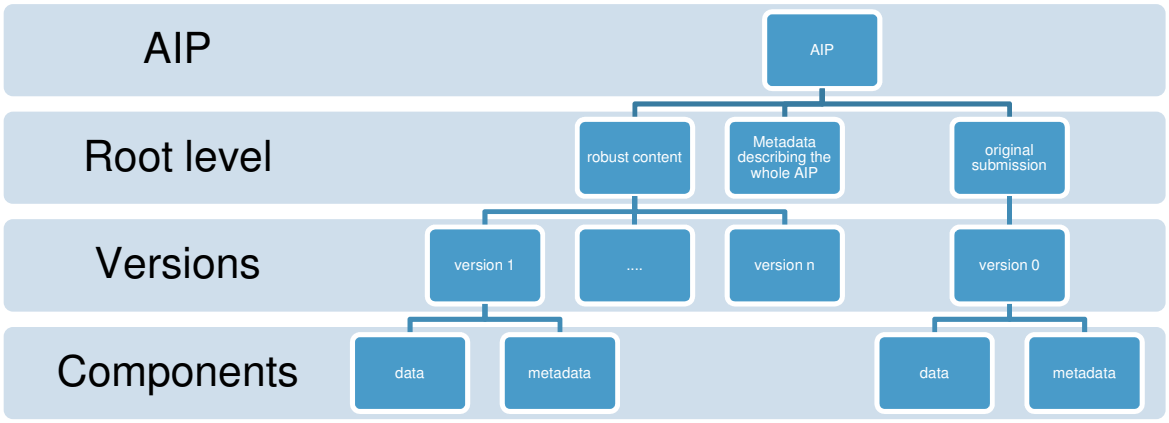

Figure 2: Compound AIP

However, due to technical limitations this might not be possible. Therefore, the underlying conceptual model can also be used to distribute the components discussed above across more than one physical container. Below we show exemplarily how this conceptual design can be implemented in different ways, but still comply with our overall design. AIPs, which contain only parts of the design shown in Figures 1 / 2 are called segmented AIPs.

The main benefit of using a segmented AIP (as illustrated by Figures 3 and 4) is that it needs significantly less physical space in the marshalling area of the processing system. The same concern for preserving space by segmented AIPs also occurs when WORM[3]-media are used. They contain parts of the AIP design, for example the *original submission* and the *robust content* in separate containers; newer versions of the robust

[3] http://searchstorage.techtarget.com/definition/WORM-write-once-read-many

content can then be stored as additional containers. Please note that the AIP containing the original submission does not necessarily contain data which are fit for preservation.

Segmented AIPs do not duplicate the original or any number of the previous versions into the new AIP, but only refer to these. Though this leads to a situation where the integrity information for the whole AIP is stored only along with the latest version (or as a standalone description as on Figure 5), it is a more cost-effective way in the short run, especially in PB-size storage situations. Nevertheless, segmented IPs can create consistency problems in the long run: so we recommend to avoid them, if funding allows.

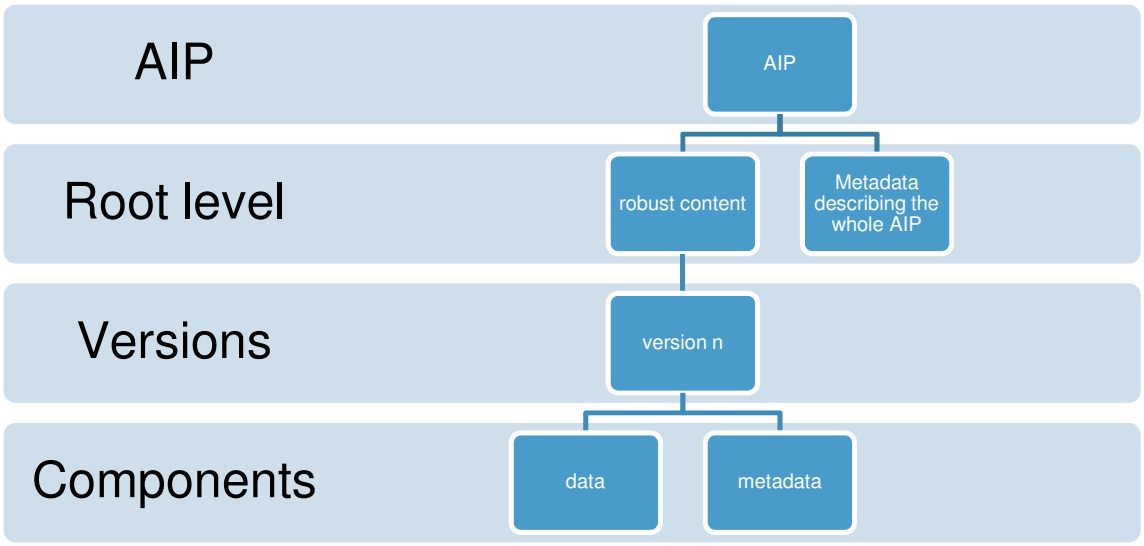

Figure 3: Segmented AIP: Migrated data in a separate container ...

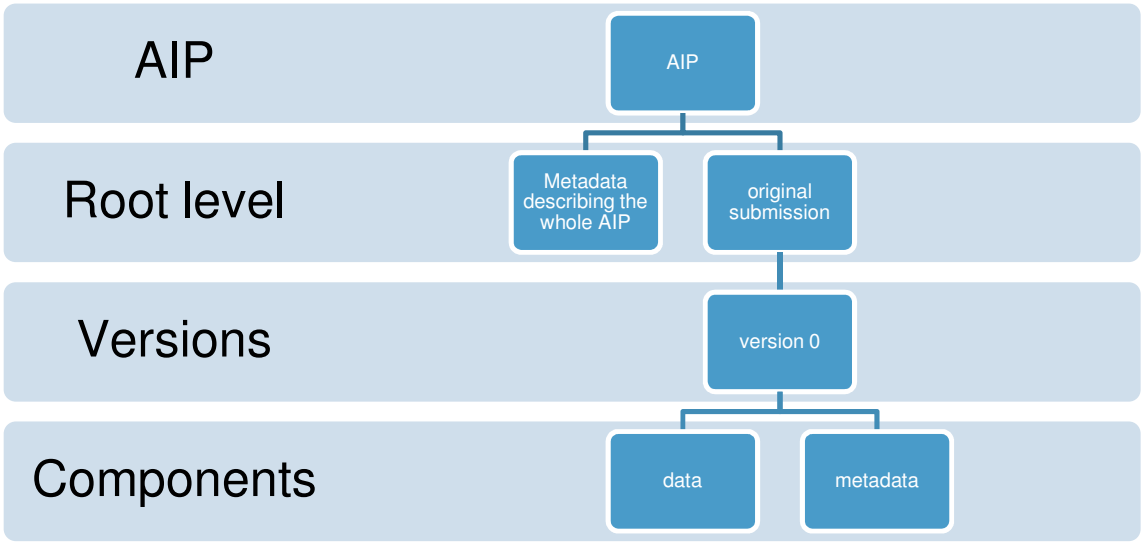

Figure 4: ... from the submitted data in another container.

In the discussion of the general case of an AIP, the AIC concept has been handled only very indirectly. For the general case, the existence of an AIC concept creates certain requirements. Basically the necessity, to create identifiers for AIPs in such a way, that an AIC can use the identifiers to describe the relationship between the AIPs. Once such a possibility exists, it may be used for a number of purposes, e.g. also to let AIPs point to other AIPs to indicate a sequence between them.

If an implementation uses segmented AIPs, however, we strongly recommend to use AICs, which contain only the *metadata describing the whole AIP* branch. The AIC then describes the relationship between the different containers representing collectively the complete AIP derived from the original submission. This is represented graphically in Figure 5.

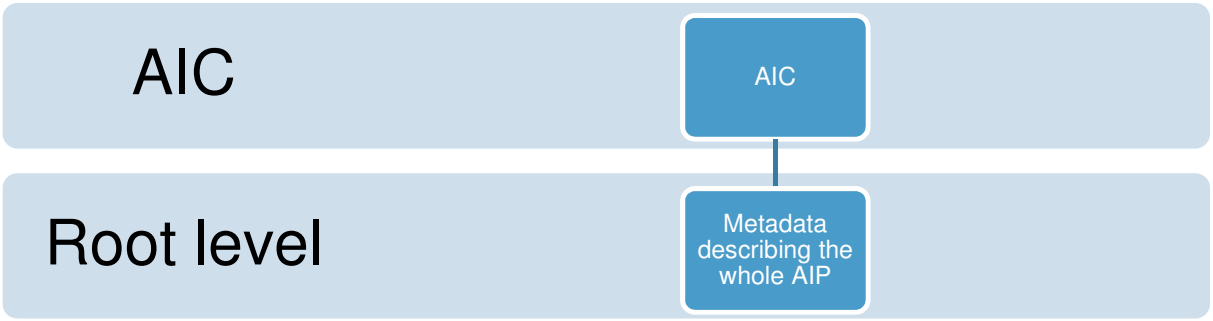

Figure 5: AIP as AIC administering relationships between segmented AIPs

In addition to this, we recommend that every segmented AIP in its *Metadata describing the whole AIP* branch provides information about the existence of this AIC, and indicates that it is only one segment of an abstract AIP. Depending on the maintenance routines an archive implements, it could be beneficial to think about storing additional information about the AIC in every AIP segment over time. This would mainly be done to prevent information loss in case the AIC is lost. (An AIP becoming unaware, that it has been logically connected to other AIPs within an AIC loses context.)

2.1.3. "Versions" and "generations".

Information packages are permanent: more precisely the information they contain is assumed to be permanent and always describing the same unaltered conceptual entity. Nevertheless, the way in which this information is represented may change. Therefore the physical representation of the information in a container may change as well.

We differentiate two ways in which this change may take place:

It may be deemed wise to have two representations, which have both been created at the same point in time. This happens, e.g., when information is submitted in a format that is not fit for preservation and will

therefore be migrated into another format during ingest. Such a migration may also happen later and be repeated. These logically different representations of the information we call *versions*.

On the other hand, the AIP may after some time be repackaged and result in another physical container. Two containers, which contain representations of the information originating from one SIP, we call *generations* of the information package.

In the case of compound AIPs, one generation of the AIP contains a number of versions of the content.

In the case of segmented AIPs, each modification to the content is stored in a new generation of the AIP.

2.2. Folder structure derived from the abstract description as a reference implementation

The abstract components of an AIP described above are represented in the container holding the AIP physically as shown in Figure 6. This is an example reference implementation which can potentially change in follow-up deliverables – e.g. by additional requirements derived from the developing Common E-ARK IP Specification.

All names for folders and files shown in dark yellow and dark grey appear exactly as shown; in the data and metadata directories, the names of sub-folders and files represent conventions that are dependent on the specific data/metadata chosen within a specific archival institution / repository system. The names shown in lighter colours represent only examples, therefore.

Names with segments in angular brackets (“<example>”), represent parts of a name generated by rules. These are the components <id>, <generation> and <version>, which are defined in the dynamic view of an AIP.

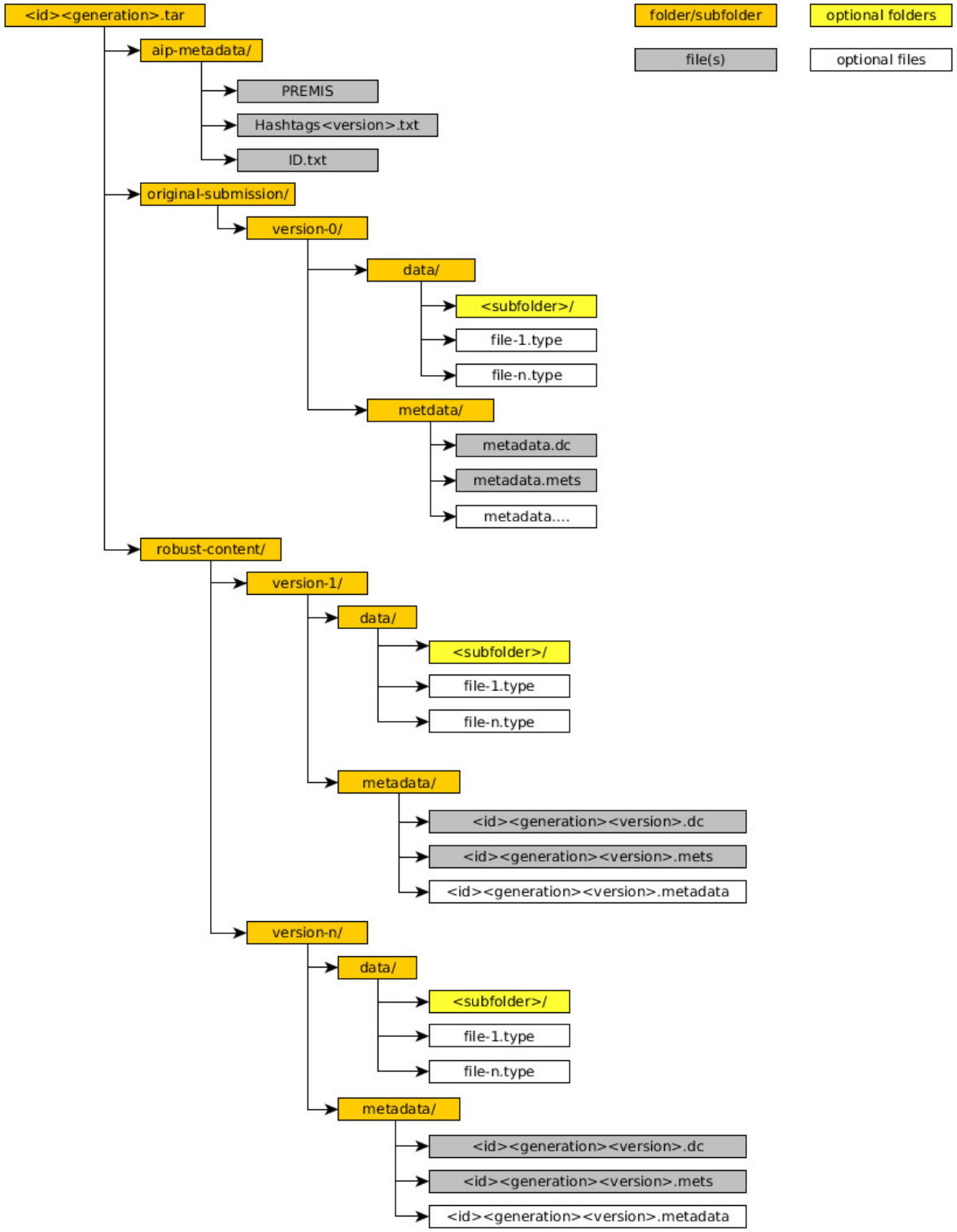

Figure 6: Concrete file structure in an AIP container

3. Dynamic View of an AIP: Lifecycle within the reference implementation

An AIP is a device to preserve an information content "forever". To be processed, the data representing that information has to be processed or rendered. As this makes assumptions about the accessibility of tools within the general IT environment, within which the IP exists, the AIP may have to adapt to changes in this environment occasionally. While the abstract information package remains unchanged, its physical representation may change.

For practical reasons a solution for another situation should also be provided. In principle, an AIP should never be changed. In practice there *will* be situations, where, after the completion of ingest of an information package, some very small section of, e.g., metadata contained in an AIP *will* change: e.g. the correction of a spelling error in the name of the creator of a multi-gigabyte video file. Keeping the AIP unchanged after this has been discovered, would imply the preservation of data known to be wrong. Re-ingesting a multi-gigabyte object, every time we discover a single item in a potentially complex metadata set to be wrong, would make the system extremely wasteful. We need an architecture, therefore, which provides three possibilities: A partial modification of an existing AIP, the replacement of an AIP by another generation and the creation of a chain of partial generations of an AIP preserving the content at various stages of change. (Such partial AIPs are referred to as *delta AIPs* henceforth.) Which of these three techniques will be selected on a given occasion is a curatorial decision. The AIP format has to support all three of these strategies, however.

To solve both problems, we envisage the following life cycle of an AIP, which defines some terms used in chapter 2 above, and defines the necessity for some of the abstract services needed to support the format described here in chapter 4 below.

The operations described below can be implemented in any repository system, which can guarantee access to an object by the following modes:

(a) Access to an object, for which we know the precise ID.
(b) Access to an object, with an ID that is the largest ID within the collation sequence used for indexing the IDs that is smaller than the submitted one.
(c) Access to the object preceding the last one accessed.

Note: *Some potential bottlenecks could be avoided if stronger assumptions are made. At this stage this specification makes the smallest possible demands on all components not covered by it directly.*

We continue describing five stages (0) – (4) in the life cycle of an AIP, where the design of the format is influenced by assumptions about the implementation of the life cycle. Generally speaking, this is a first iteration of the reference model which is subject to change.

(0) An SIP will usually be converted into exactly one AIP. This is also the only model that will be supported by the first version of the SIP2AIP converter. Future versions will be required, however, to support an "assembly stage" (to create segmented AIPs and to handle segmented SIPs).

(0.1) During this assembly stage, very large SIPs may be broken down to a set of "transient SIPs". A transient SIP will always contain a set of consistent data and metadata (all data present to be documented by the metadata; all data mentioned in the metadata actually present). Some of these transient SIPs may be converted to SICs (Submission Information Collections). If this is the case, the resulting transient SIPs will be analyzed during creation. SIPs which are independent of other SIPs will be converted to AIPs first, as described in item (1). Therefore, for the remaining SIPs (possibly: SICs), the AIP IDs, which represent the SIPs to which these SICs refer, will be available at the time the SICs are converted into AICs. If deadlock conditions are discovered (SIC 'a' referring to SIC 'b', when SIC 'b' refers at the same time to SIC 'a') a transient SIP may be assigned a preliminary AIP ID. Then this AIP ID has to be flagged as preliminary in the ID.txt file of the AIC which refers to it.

(0.2) During the assembly stage multiple SIPs may also be combined into one AIP. As this complicates the process described in item (3) below, it is currently deemed unwise to implement this operation, unless important specific reasons apply.

(1) When an AIP is created during ingest, it receives an unalterable identifier, which defines the AIP as one consistent logical entity. This identifier *must* consist of a prefix, which reflects the structure of the IPs delivered by the institution from which it has been submitted, and it must be closed by a running number. This ID is referenced as <id> in Figure 6. We *recommend* using as a prefix an internationally recognized standard identifier for the institution from which the SIP originates. This may lead to problems with smaller institutions, which do not have any such internationally recognized standard identifier. We propose in that case, to start the prefix with the internationally recognized standard identifier of the institution, where the AIP is created, augmented by an identifier for the institution from which the SIP originates.

An initial premis.xml file is created, which describes all events up to and including the deposition of the AIP in the repository.

If necessary, because the formats in the version-0 are seen as unfit for preservation, data and / or metadata can be migrated to form a new <version>. That is a complete representation of the original data or metadata contained within the *original submission/version-0* branch of the AIP in a new format, with the string "version-nn" as folder name. "nn" is an incremental number. The fact of the creation/availability of these versions, as all other relevant details of the conversion, are stored in the premis.xml of the AIP.

Hashtags are computed for the version-0 branch and any additional versions created and stored in one or more Hashtags-<version-number>.txt file. Each of these is representing one version and becomes part of the *aip-metadata* branch.

The components of the *aip-metadata* branch, the *original submission* branch and such *versions* as have been created are packaged as a *robust container*, which has as its name the basic ID, suffixed with a <generation>-number 0 (zero).

This *robust container* is sent to the repository.

(2) As a response to a request from the technology watch realized within the repository containing the AIP, data or metadata can be migrated to form a new *version*. That is a complete representation of the original

data or metadata contained within the *original submission/version-0* branch of the AIP in a new format, with the string "version-nn" as folder name. "nn" is an incremental number. The fact of the creation/availability of the *version 0*, as all other relevant details of the conversion, are stored in the premis.xml of the AIP. If possible, the Original Submission should be used, to avoid data loss. The technology watch mechanism is not part of this specification: this item describes the implications of such a technology watch for the format, however.

For this purpose the *version* with the lowest version number, which contains a format that can still be processed, is extracted and converted to the new format, unless a specific request to base the migration on a specific version in the AIP has been submitted.

Any relevant information on the migration is added to the premis.xml.

Hash values are computed for the components of the new version and stored in a Hash value-<version-number>.txt file.

The updated *aip-metadata* branch, the unaltered *original submission* branch and such *versions* as are to be retained by the migration request are packaged in a new *robust container*, which has as its name the original ID with a new generation number incrementally assigned.

This robust container is sent to the repository.

A request to delete the robust container with the previous generation number *may* be sent to the repository. Whether the repository reacts with a physical deletion, or - particularly in the case of WORM devices - with a logical deletion (the older generation being excluded from further low level transfer to new generations of hardware) is no concern of the AIP.

(3) An institution from which a SIP has been submitted, may request a modification of an AIP. Our assumptions about the repository do *not* cover multiple identifiers for one object. This implies that a depositing institution is able to store the AIP identifiers sent back as acknowledgement after a successful ingest operation. This is a potentially severe restriction in the handling of transient SIPs, as described above under items (0.1) and (0.2).

A modification request submits a partial SIP, which may contain one of the following:

(a) A complete set of metadata replacing the *metadata* branch previously submitted.
(b) A set of data files partially replacing data files already submitted.
(c) A set of data files augmenting the already submitted ones.
(d) A new PREMIS section describing any access restrictions / rights stored within the premis.xml file of the AIP.

The modification request leads to the creation of a *delta AIP*, which has a generation number incremented beyond the highest generation number of the AIP in use so far. This AIP is deposited in the repository.

(4) The logic described in (2) and (3) assumes, that an AIP being required is retrieved according to the following logic. This will have to be implemented in part by the AIP2DIP conversion tools to be implemented within Work package 5 of E-ARK.

If an AIP is requested, where the ID submitted contains a generation number, the AIP with this generation number is delivered, if existing.

If an AIP is requested, where the ID is submitted without generation number, the AIP is retrieved according to the following rules:

(a) The AIP with the basic ID and the highest generation number is retrieved. If it is complete, it is delivered.

(b) If this happens to be a *delta AIP*, its components are extracted into a transient package. Then the AIP with the next lower generation number is retrieved and its content is used to extend the transient package. "Extending" means that only such components are integrated into the transient package for which no representation has been found in the previously retrieved generation. This is continued, until a complete AIP is encountered from which the transient package is completed, before being delivered.

4. Generic API of a SIP2AIP supporting E-ARK AIPs.

4.1. Abstract description

As mentioned initially, we cannot discuss the AIP format in the context of E-ARK, without covering the implications it has when embedding it into the E-ARK structure as a whole. In D4.1: "Report on available formats and restrictions" the vision of a Pan-European AIP format has been described as a catalogue of requirements on pp. 9-10. Its role in the overall workflow, supported by two converters, has been visualized by Figure 7. In the current chapter, we describe a few general conclusions we have derived from the format proposed above for the implementation of these converters.

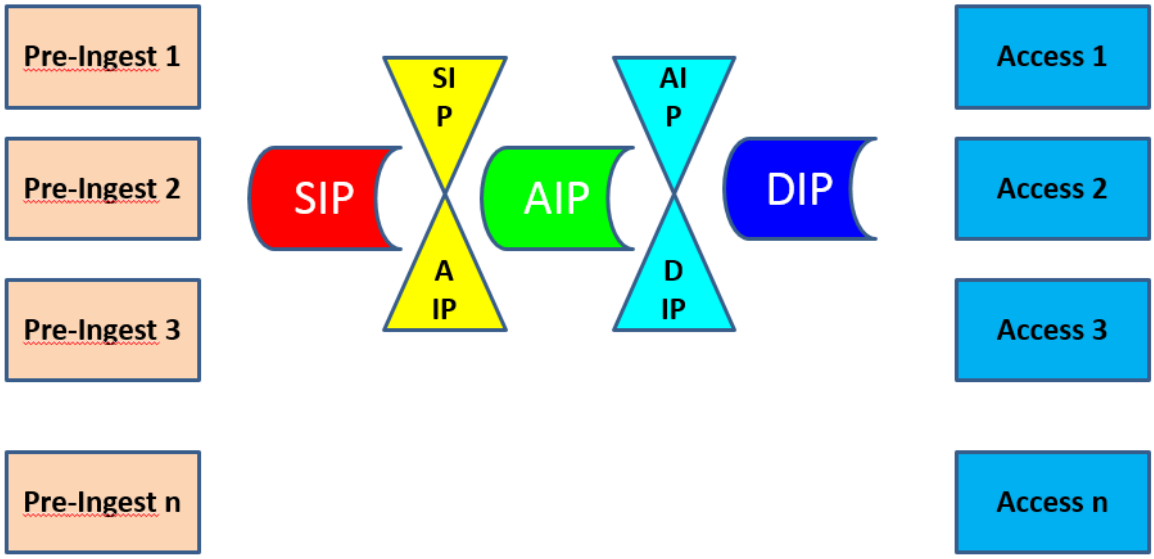

Figure 7: Vision of a Pan-European AIP format

As the Common Specification - the general concept of an E-ARK IP – is at the time of writing still under discussion within E-ARK, appendix I describes the understanding of the specific properties of AIPs in the context of IPs in general, which has governed this document.

As the SIP2AIP converter is required to structurally support any formats required by the depositing institutions, we expect the converter to have the following overall structure.

A generalized framework program is created, which is aware of the Pan-European SIP format and directs the general flow of execution. This general flow of execution directs, e.g., the application of the final *robust container* and it decodes flags describing the operation as a whole (e.g.: "this is a *delta AIP*"). It gets its power and generality by providing the possibility to "register" components, which are to be called, if a specific operation is requested. Most of these operations are specific for a single format or pair of formats. At the time of this first draft we have identified the following ones:

(a) Create an E-ARK ID according to the requirements of a specific implementation.

(b) Identify a metadata format.

(c) Validate a metadata format.

(d) Identify a data format.

(e) Validate a data format.

(f) Check consistency of (A(X), B), where A is a set of metadata in format (X) and B is a set of data objects.

(g) Enquire about fitness for preservation of metadata format X, where X is a format identifier, returning a target format for conversion, if required.

(h) Enquire about fitness for preservation of data format X, where X is a format identifier, returning a target format for conversion, if required.

(i) Convert metadata format (f, X, Y), where f is an individual file and X and Y are format identifiers.

(j) Convert data format (f, X, Y), where f is an individual file and X and Y are format identifiers.

(k) Apply hash to F, where F is a set of files contained within the AIP.

Any component has to support a callback that returns format specific error conditions.

At least components of the type (i), (j) and (k) have to support a callback that returns a valid PREMIS description of the action taken.

Note: *Logical components not mentioned here as software components – e.g. the mechanism of how the failure of a specific operation is communicated to the submitting institution – are considered to be part of the framework, which calls upon the software components specified here as needed. The components here are to be supported by an architecture, which allows them to be plugged in, without changing the overall framework.*

4.2. Interaction between AIP format properties and API requirements

Within work package 4, the main work running parallel to this specification has been the implementation of a preliminary version of an SIP2AIP tool, which has been integrated into the general demonstrator implemented at the Austrian Institute of Technology, as a part of E-ARK. This realizes the very basic workflow of an SIP2AIP conversion tool and is intended to check the feasibility of the structures described above. It will also allow the implementation of further refinements of this format specification, concurrently with the design and implementation of the SIP2AIP converter supporting it.

The current stage of the preliminary implementation is shown in Figure 8. This is a description of the current implementation, which has been implemented for the purpose described in the preceding paragraph. It neither claims to support the full use case elaborated within work package 2 at this stage, nor does it consider a clear separation between pre-ingest as handled within work package 3 and ingest, as handled by this work package.

The decision, whether the individual file is fit for digital archiving, is currently hard coded into the software. This part of the preliminary converter will be replaced by a call to the registered component “Enquire about fitness for preservation of format (X)”. Similarly, the format migration is currently hard coded and will be replaced by the registered component “convert format (f, X, Y)”.

This means, that the SIP2AIP converter is implemented as a framework creating AIPs according to the format description given above and substantiated in Figure 7. The “operational semantics” – i.e., the decision about which formats are preferred or which tools for the actual format conversion are trustworthy – can be instantiated differently for each archiving institution, according to different institutional policies. While no

decisions about concrete technologies have been made at this stage, it is clear that most of the registered components will be lightweight wrappers around existing tools.

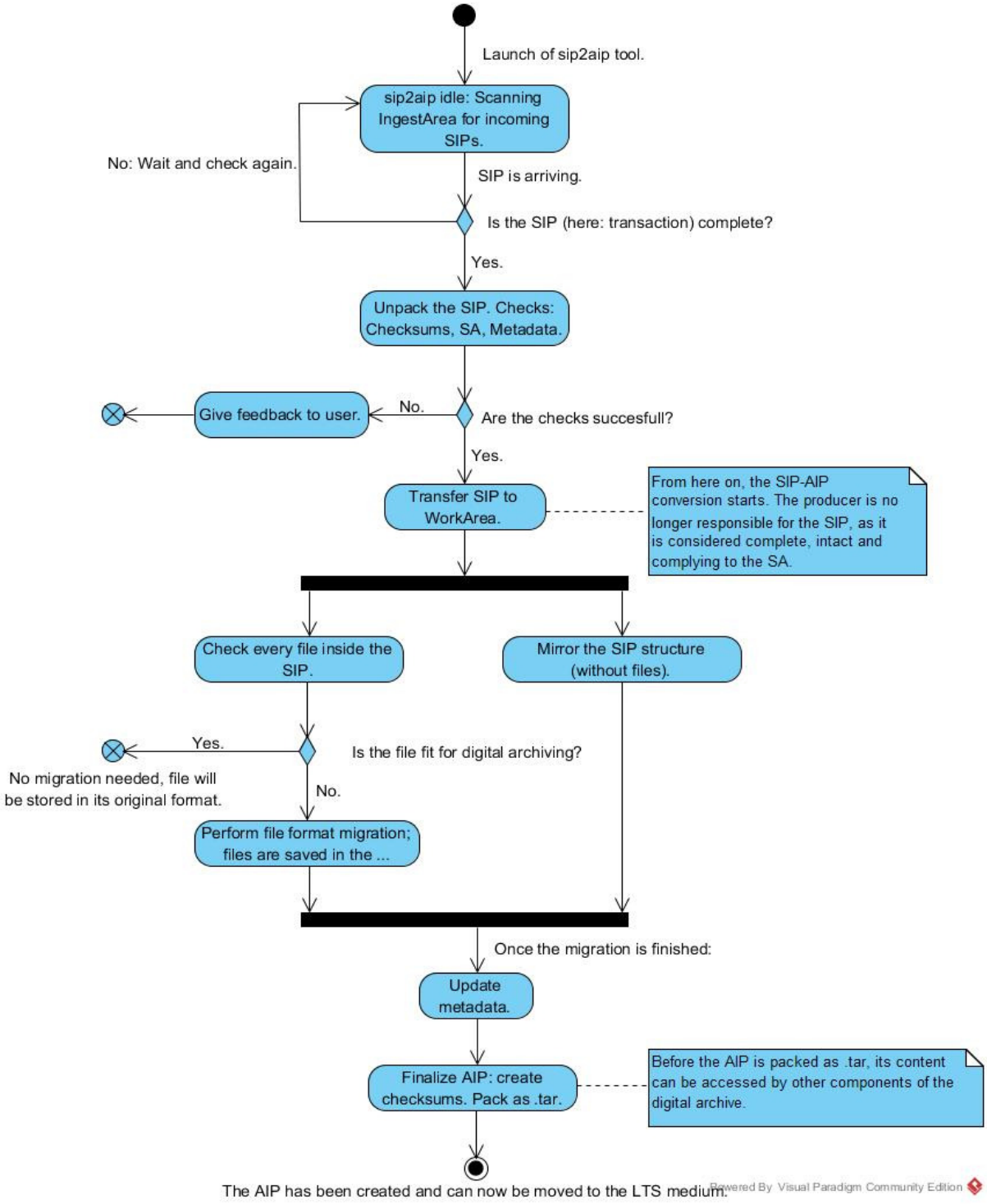

Figure 8: SIP2AIP Overview

Appendix I – Assumptions about the Properties of Information Packages Underlying this Document.

As the general concept of an IP – as a common ground for SIPs, AIPs and DIPs – is at the time of writing still under discussion within E-ARK, we provide here the understanding of the specific properties of AIPs in the context of IPs in general, which has governed this document.

(1) Assumptions about Information Packages in general

An information package – IP - is a digital object, which can be transported between information systems as one continuous byte stream that contains data and such metadata, as are needed for a specific purpose.

An IP has different degrees of dependency on a specific processing environment. This can also be expressed as its degree of autonomy, or independence of a specific processing environment.

A processing environment consists of the hardware and software on which an IP is processed, and of the brainware of the user of these components.

A single file in a specific format – TIFF, DOCX, PDF … - is an example of an IP, that is extremely dependent, both on the technical part of its environment, as well as on the brainware.

The lower the dependency of an IP on a specific processing environment, the greater its autonomy, the fitter it is for preservation.

There are no technical solutions for the operation of the brain of the human user. Therefore, we need to distinguish clearly between: (a) The parts of an information system, which depend on the direct application of any human knowledge which is needed for the content aware operation of the system. (b) Those parts which are "content agnostic", which can be implemented without any knowledge of the specific content of an IP, as long as all the dependencies on hardware and software are known.

(1) Consequences for Information Packages within preservation systems

There should be a clear distinction between those parts of an ingest process, which are content aware and those parts which are content agnostic. As OAIS was developed as a general design for all archives – the spirit of OAIS would assume, that medieval parchment charters should also be administered by an application of OAIS – there is no clear distinction between content aware and content agnostic technical operations.

When designing the E-ARK AIP format, we assume:

(1) A conversion component which converts a Pan-European SIP into a Pan-European AIP has to be independent of any local regulations. It must operate completely on the content agnostic level. This is

only possible, if the ingest process is organized in such a way, that there exists a stage, where the IP can be processed fully automatically. An IP at this stage is what constitutes an SIP from the point of view of the work package responsible for AIP handling.

(2) The task of SIP2AIP conversion consists of a technical layer, which is responsible for ensuring a unified quality and interoperability of AIPs. It therefore performs tasks like format validation (as far as that concept makes sense), consistency checks, the addition of fixity information and the various other tasks implied by D4.1.

(3) An AIP has to be as generic as possible. I.e.: All processes which generate it, have, e.g., to support "METS", rather than a specific subset of fields of METS. An AIP definition has therefore in principle to decide which metadata standards are supported, not which fields of them.

(4) A concrete implementation of a SIP2AIP conversion will obviously not be able to support all existing metadata or data formats. The responsibility of the design of the AIP format and the SIP2AIP converter is to ensure, that it is as easy as humanly possible to add the support for additional formats. The responsibility of the implementation of the E-ARK SIP2AIP conversion is to make sure, that those formats, which are required by the archival partners, are actually served.