www.forgetit-project.eu

ForgetIT

Concise Preservation by Combining Managed Forgetting and Contextualized Remembering

Grant Agreement No. 600826

Deliverable D5.2

Work-package	WP5: Joint Information and Preservation Management
Deliverable	D5.2: Workflow model and prototype for transition between active system and AIS - first release
Deliverable Leader	Jörgen Nilsson
Quality Assessor	Mark A. Greenwood
Estimation of PM spent	9
Dissemination level	PU
Delivery date in Annex I	M12
Actual delivery date	2014-03-17
Revisions	7
Status	Final
Keywords:	preservation workflows, ingest, access, recontextualization, CMIS, integration

List of Authors

Partner Acronym	Authors
LTU	Ingemar Andersson
LTU	Parvaneh Afrasiabi Rad
LTU	Göran Lindqvist
LTU	Jörgen Nilsson
LTU	Tero Päivarinta
IBM	Simona Rabinovici-Cohen
DFKI	Heiko Maus
dkd	Olivier Dobberkau
EURIX	Walter Allasia
EURIX	Francesco Gallo

Contents

Executive summary

This report is the first in a series of three which will document the Integration approach as well as the Archiver component and the Context-Aware Preservation Manager produced by WP5 for the ForgetIT project. The components described here are initial prototypes, and concern the Context-Aware Preservation Manager which is still under discussion and which will continue to be developed over the following two deliverables. This work builds on concepts and ideas developed and documented in D5.1.

Glossary

List of important terms and acronyms presented in the document:

AIP	Archival Information Package
AIS	Archival Information System
CMIS	Content Management Interoperability Services
DIP	Dissemination Information Package
GUID	Globally Unique Identifier
METS	Metadata Encoding and Transmission Standard
MODS	Metadata Object Description Schema
PAIMAS	Producer to Archive Interface Methodology Abstract Standard
PoF	Preserve or Forget (Middleware)
PREMIS	Preservation Metadata: Implementation Strategies
SIP	Submission Information Package
XML	eXtensible Markup Language

1 Introduction

One of the purposes with the ForgetIT project is to provide a foundation for synergetic preservation, where information management and preservation management has more vivid interaction, making preservation a more integral part of content management in both organisational and individual contexts.

As identified in ForgetIT deliverable 5.1, there exists a gap between Enterprise Content Management Systems and Preservation Systems, especially if, for example, we consider automated processes for ingest. To point out some more explicit points, we can look at what Korb & Strodl [1] identified as gaps between Enterprise Content Management Systems (ECM) and OAIS:

- ECM provides no preservation planning functionalities, leaving them to be taken care of by the Preservation Planning and Administration functions of OAIS.
- In ECM records management, migration has been mostly in the form of migration of data from one storage medium to another, not so much about migrating file formats when they have become obsolete due to changes in an ECM.
- ECM systems collects information produced by an organisation. The OAIS however, needs to be provided with the information that is to be preserved.
- ECM systems are mostly integrated to the organisational infrastructure while Preservation systems often are external services provided by external organisations. This could lead to a lack of alignment.
- The ECM gathers metadata (sometimes automatically) about content ownership, access rights, and other organisational issues related to context and the active part of an object's lifecycle. The preservation system is more specialised in preservation.
- The preservation system (OAIS) need to store descriptive metadata separately from the actual content object. In ECM these are usually tightly coupled.

If we also add to this the low penetration of digital preservation practices in industry and that organizations are often not wiling to allocate a lot of resources on the preservation activities [2], and the aim for ease of use and seamless transition of information between Information Management systems and Preservation systems, then we have ourselves a challenge. Let us first start by looking at some high-level workflows for information management and digital preservation.

1.1 Structure of report

After the Introduction, a section describing circumstances related to High-level Workflows and Integration Considerations follows. The report then continues with describing current workflow descriptions for ingest and recontextualization. Section 4 describes in some detail the implementation done so far on components and middleware related to Workpackage 5, followed by Conclusions and Future Work. Appendix A and B contains class descriptions of the components from section 4.

2 High-level Workflows and Integration

This section describe high-level workflows related to *Information Creation*, *Pre-Ingest and Ingest*, *Access*, and *Preservation Planning*. The focus lies on the purpose of establishing seamless transition of information objects between Information Management systems and Preservation systems, which also entails integration of the systems. Therefore the latter parts of this section looks at general integration approaches and describe the choice made in the project.

2.1 Workflows

It is worth remembering that the particular workflow for a specific case will be highly individual, especially regarding the *information creation*, and therefore these descriptions are on a generic level. The generic level should be useful to spot common choke points and problems that may arise in order to address these in as seamless and transparent a way as possible.

2.1.1 Information Creation

Since the project targets individuals and organisations that generally do not have archiving expertise, it is not possible to put up harsh requirements on what has to be done regarding metadata creation during the actual information creation (i.e. the creation of the digital object which later will be preserved). In some contexts, such as in the case of TYPO3, the system automatically attaches metadata such as time of creation, author, title, and similar to the object, which means that some metadata exists. What has to be done is to make sure that we can get the metadata out of the producing system into the preservation system. This is mainly a task for the Pre-Ingest and Ingest workflows, but also involves Preservation Planning. Additional metadata, not provided by the producer, will be added through processing in the ForgetIT middleware, e.g. by the *Contextualizer component*.

2.1.2 (Pre)Ingest

On a high level, the *Producer to Archive Interface Methodology Abstract Standard (PAIMAS)* [3] describe what needs to be covered during establishment of a “contract” between a *Producer* and an *Open Archival Information System (OAIS)* [4]. This involves 86 steps divided into four phases for covering the transfer of a digital object from the producer to the preservation system. The four phases are:

- Preliminary phase (46 steps)
- Formal Definition Phase (36 steps)

- Transfer Phase (2 steps)
- Validation Phase (2 steps)

These phases are depicted in figure 1 where also a brief description of the phases can be seen.

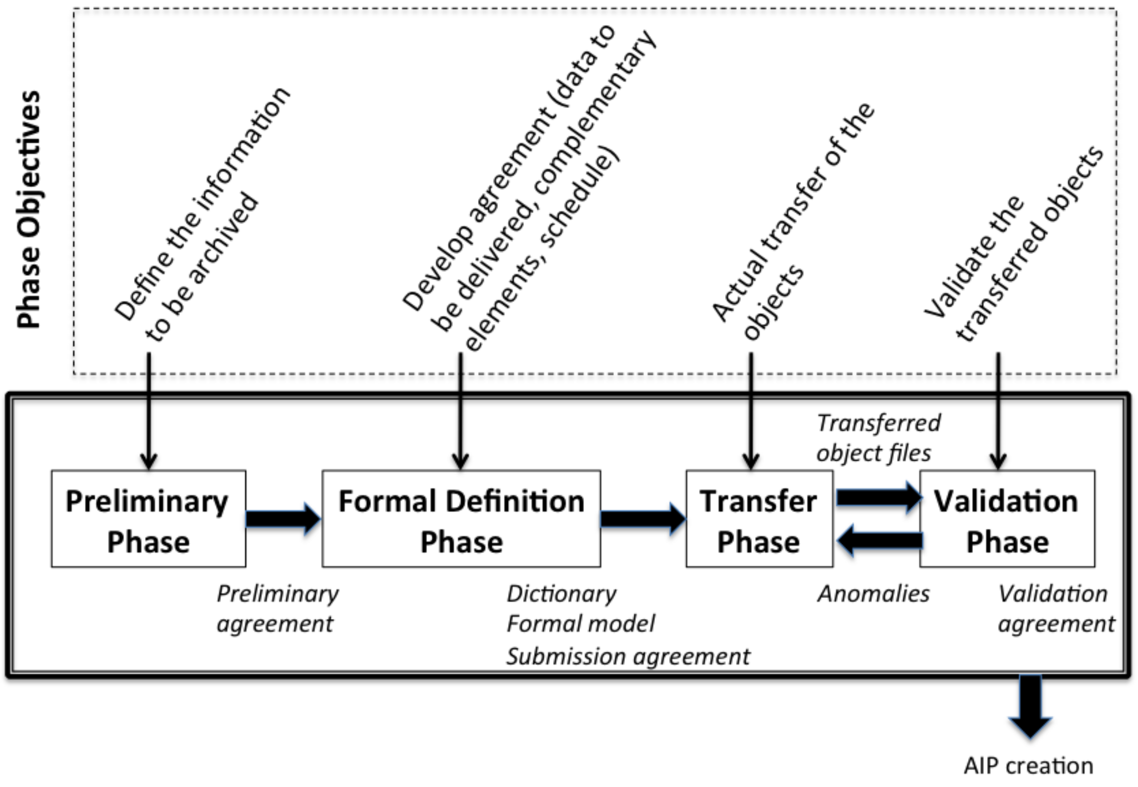

Figure 1: PAIMAS Main Phase Objectives and Outputs (adapted from [3])

Already the sheer number of steps, let alone the work required in some of the steps, makes it problematic for people not accustomed to records management and archiving processes. Since this project specifically intends to assist organisations and individuals that are unfamiliar with the formal processes involved, just giving them the PAIMAS model to implement is not feasible as highlighted by Keitel [5]. The PAIMAS model together with the OAIS model is, however, a good foundation for identifying steps that can be automated and information that needs to be collected, preferably in an automated way.

2.1.3 Access

What in OAIS terminology is called “Access” in the ForgetIT project also includes the act of recontextualization where the purpose is to get the information into a different system than the original one, retaining important context and other properties (e.g. access rights). A typical OAIS access scenario, illustrated by the Access Functional Entity (figure 2) would include: searching/querying for interesting information objects where the OAIS would return descriptive information for relevant objects; selecting and ordering the actual objects that the customer wants; getting the objects delivered in one or many Dissemination Information Packages (DIP).

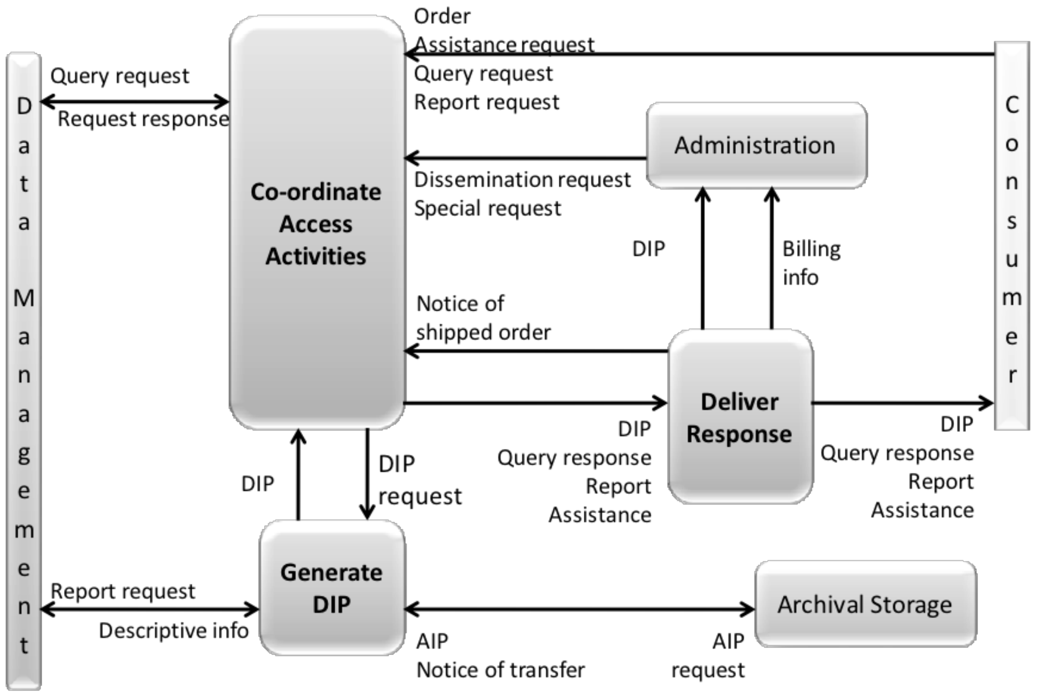

Figure 2: OAIS Access Functional Entity [4, p. 4-16]

In this project, the Preserve-or-Forget Middleware will be the mediator between Consumer and the OAIS, and also provide some extra search facilities (especially regarding contextual searches) (figure 3).

2.1.4 Preservation Planning

In the OAIS model, Preservation Planning is one of the functional entities modelled. The main purpose of Preservation Planning is to make sure that information objects preserved in the system actually remain usable and understandable. In order to do this, it is suggested that it should work with monitoring of the *Designated Community*, which is OAIS terminology for “target group”, in order to better serve the users according to their needs and knowledge base, which both may change over time. This also means that descriptive information and context information need to be updated, in order to accommodate the change in the Designated Community [4].

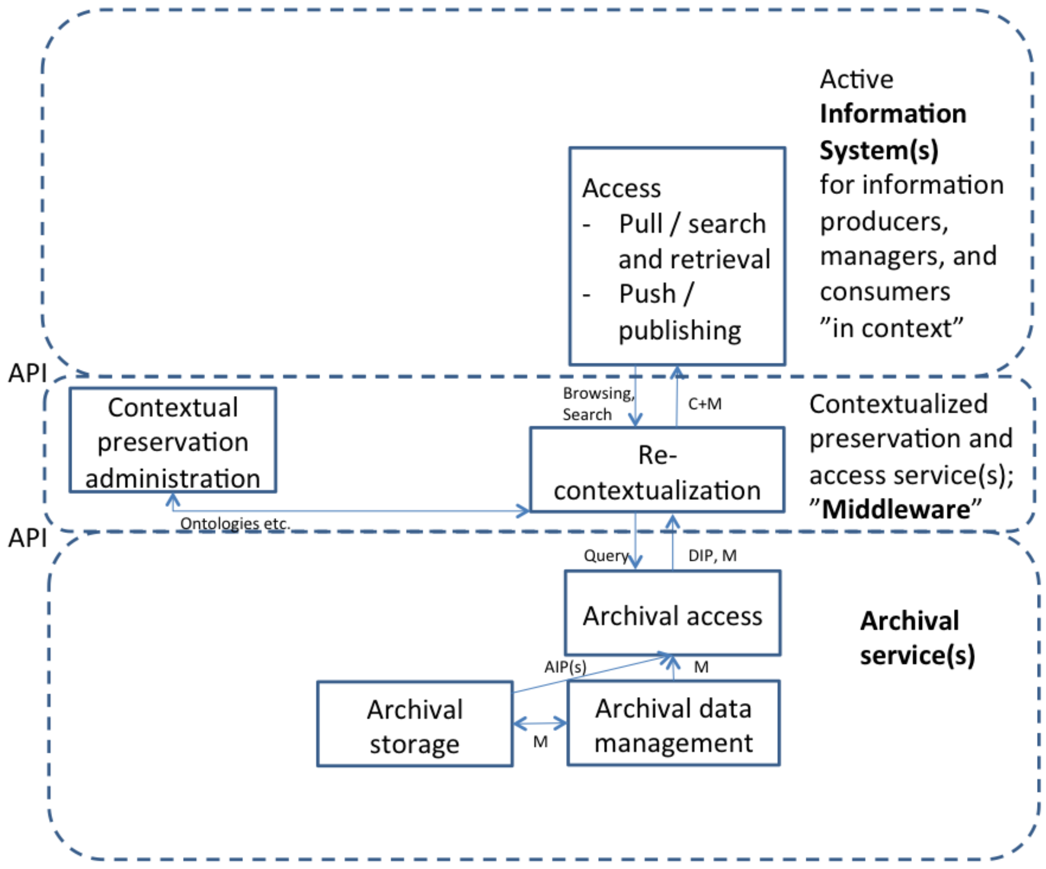
Active Information System(s) for information producers, managers, and consumers "in context"
Access
- Pull / search and retrieval
- Push / publishing
API
Contextual preservation administration
Browsing, Search
C+M
Re-contextualization
Contextualized preservation and access service(s); "Middleware"
Ontologies etc.
API
Query
DIP, M
Archival access
Archival service(s)
AIP(s)
M
Archival storage
M
Archival data management

Figure 3: Abstract elements of retrieval and access workflows (from D5.1)

Preservation Planning also involves monitoring changes in technology and planning for how to tackle those changes. If, for example, a file format gradually becomes less and less supported by newer software and systems in general, it is necessary to plan for how to deal with it (especially if the AIS has large amount of objects in that particular format). In order to bring the Preservation Planning function of the OAIS closer to the producers and consumers in the ForgetIT context, we therefore suggest supporting it by implementing a *Context-Aware Preservation Manager* function in the PoF middleware. This "manager" will coordinate preservation oriented information exchange between the Producers, Consumers and the AIS. This component has not yet been implemented, but will be developed during year 2 and year 3 of the ForgetIT project.

2.2 Integration Considerations

In the following we briefly discuss the approach chosen for integration of the active systems with the AIS. The information exchanged between the active systems and the AIS are of different character, since we e.g. have searches, ingest requests, actual transfer of objects, and triggers of events such as obsolescence of a certain file format in the AIS. Besides this we also have internal processing in the PoF middleware that might take considerable time, such as image analysis and contextualization. As yet another complication, the chain of events differ depending on type of request, and type of information objects concerned in the exchange. Many of these concerns and considerations have been thought of during year 1, but the answers to what is preferable or more suitable will come when the 1st iteration of the prototype system is up. Some examples of what has been under discussion is:

- What is the granularity of the preservation?
  Choosing whether to preserve single files or collections of files is essentially up to the Producer to decide upon. However, handling many single file AIPs in the Preservation System might be a problem in the long run and the Preservation System might decide to combine objects of similar type/origin/context into larger AIPs.

- Who triggers the preservation?
  In general, the Producer triggers preservation, but the decision should be supported by calculations of Preservation Value done by the Forgettor component. Based on this, decision could also be made by the producer side system to trigger preservation automatically - but for Y1 the preservation is triggered manually. Some examples of “trigger points” in the active system could be; when an object is published, when a certain already published object reaches high popularity, when a user decides actively that an object should be preserved, when an object is about to be deleted or removed from active system.

- Context information in the active system?
  Objects might, and probably should, have some context information already in the active system (Producer system). How to best utilize relevant parts of this and embed it in suitable metadata schemas is something that we intend to investigate further.

- What is the interaction between versioning and preservation?
  This is something that has been discussed, but not “resolved”. A preservation system should not be considered as a “backup” system, or a “version control system” - but it should however support versions of the same object. One of the reasons for that is that the preservation system itself might decide to migrate objects to other file formats, but also because the producers might provide a corrected version of something that already has been preserved and that needs to be kept for reference (e.g. in some legal context).

All in all, some of these questions are in focus in the project, such as “context information” while others are interesting to discuss albeit something that does not have to be resolved in the scope of the project. The latter type will be discussed and implemented in a more pragmatic fashion in order to have a working prototype system, while the first type of questions will be more extensively scrutinised to have a good foundation for deciding upon what to do and how to do it.

3 Workflow Descriptions

This section describes in more detail the workflows involved in implementing the seamless transition of information objects from active system to preservation system and back again. At this stage it mainly concern Ingest and Access, while the Preservation Planning functionalities will be dealt with later in the project.

3.1 Assumptions

The work described here is based on a number of assumptions that are decided upon in the project as a way of limiting the scope of what needs to be implemented while still serving as a base for the research to be conducted in the project. One such assumption is that the active systems make their resources available via CMIS[1]. This is mainly a pragmatic choice in the project based on that one of the systems is a Content Management System, the intended adopters of CMIS. In a fully implemented system for production use, other branch standards would be considered for adoption. Another assumption relevant to describe here is that most components in the project, including those mentioned below, are implemented in a middleware with loose couplings between the components. A third assumption is that the Preservation System in this case is a DSpace installation, and that few changes should have to be made to the Preservation System side other than for communication and storage (storage solution is dealt with in WP7).

3.2 Pre-Ingest and Ingest

During the Pre-Ingest and Ingest phase (Figure 4), information is first targeted for preservation in some way by the active system, either automatically or deliberately by some user of the active system. The latter alternative is not the main focus of the project, but certainly a scenario that might occur depending on context. After the information has been selected for preservation, the active system notifies the Preserve-or-Forget Framework with a *Preservation Request*. This request is forwarded by the message broker in the middleware to the *Archiver* component. The Archiver then extracts the CMIS-GUID (a global unique identifier) from the request and notifies the *ID-Manager* via the message broker in order to establish an identity to use for the information package.

The Archiver now fetches the CMIS Content from the CMIS Repository pointed out by the request, and puts this in a shared file area for processing using the GUID as identifier (i.e. name) of the catalogue. When the transmission is done, and verified, the Archiver notifies the message broker that new information objects now are available for internal processing in the middleware. In the figure, the *Contextualizer* is pointed out, but other components

[1] Content Management Interoperability Services. `https://www.oasis-open.org/committees/cmis/`

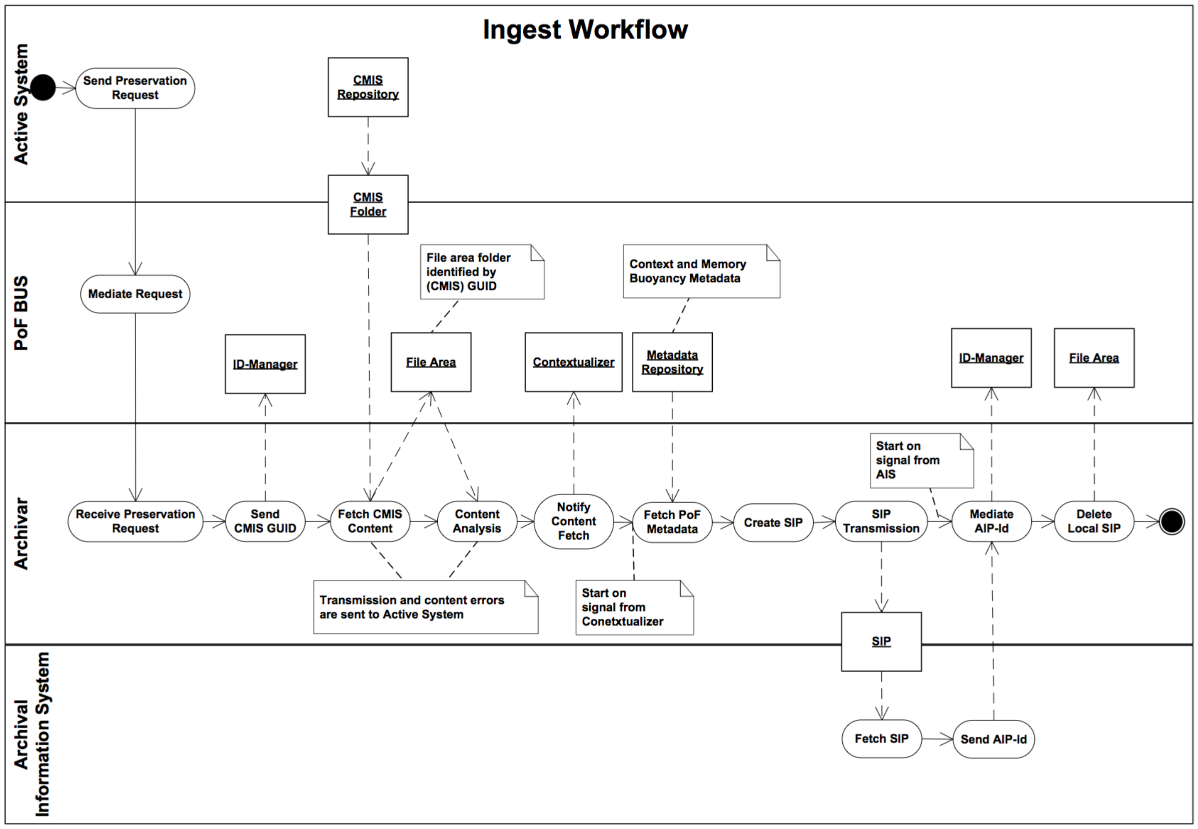

Figure 4: Ingest Workflow

will also process the information objects. When they do, the metadata they extract and produce go into the share metadata repository either as files or as entries in a database, sometimes even a combination of both.

When all relevant middleware components have finished working on the objects, the Archiver component is notified by the bus and commences packaging of the objects and metadata into a *Submission Information Package (SIP)* (figure 5). At the moment the SIP is created according to a DSpace-METS schema, since the project is using DSpace as preservation repository. Some of the metadata extracted earlier go into the METS schema, or subsection thereof (e.g. the MODS section) while some more specialised metadata from the PoF processing is packaged as metadata files in a subfolder labelled “metadata”. The actual content objects reside in a folder labelled “content” and there is also a folder labelled “system” which can be used for holding objects related to execution or presentation of the content objects. The SIP is then served to the Archival Information System (AIS)/Preservation System and when the AIS successfully have received the package, an identifier for the Archival Information Package is sent back to the Archiver component, which then notifies the ID-Manager of this. When the package has been ingested into the AIS, both the package and the source files (i.e. content objects and metadata) can be removed from the file area in the middleware.

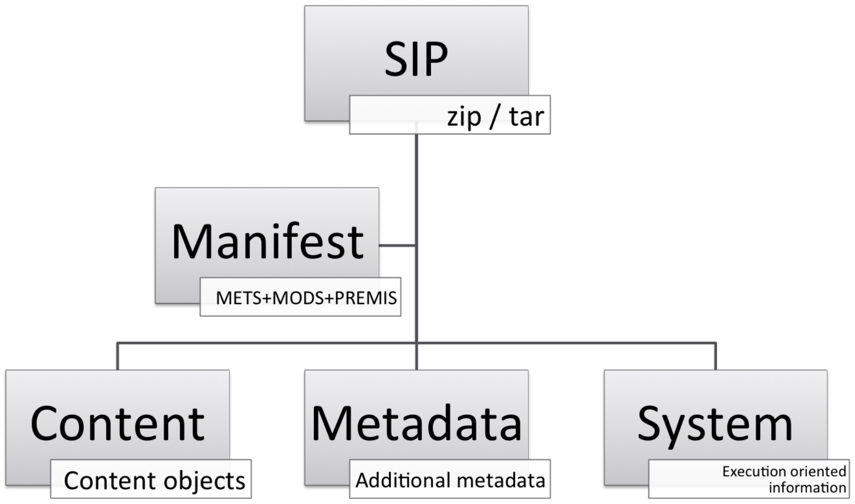

Figure 5: Submission Information Package

3.2.1 Communication

In the project we work with the CMIS protocol for exchanging information objects with the active systems. The process relies on having GUID available for the objects, and will either receive such from the active system, or in lack of such generate identifiers in the middleware.

The preservation system on the other hand, does not handle CMIS and therefore the objects are packaged according to the METS standard in preparation for ingest to the preservation system. After successful ingest into the AIS, we store the AIP ID in the middleware ID manager for future reference.

3.3 Access and Recontextualization

While “access” is a word used in OAIS, and generally a graspable concept, the ForgetIT project also strive for “recontextualization” of preserved information. This entails accessing information objects that originated in one system and putting them into a sensible, but perhaps different, context than the originating context. This would e.g. include changes in the actual information system environment (different software) but also changes in the use of the object, or changes in the underlying information model. All in all the purpose is to facilitate use of the information object, in a way that fits the purpose of the future user.

Even though recontextualization is the goal, we do need to get basic access working to begin with. The workflow in figure 6 describes the first iteration of this functionality.

The active system sends an access request for a specific object. This request is mediated through the PoF bus to the Archiver component. There are at least two ways for the active system to know what to look for. It could either be that they have the GUID in their own records, or they could use PoF search functionality. This is however not depicted in the workflow since it deals with the fetching and delivery of an information object. When the request is received, the Archiver component contacts the ID-manager (if necessary) to get the AIP ID needed for the request to the AIS. The AIS is then contacted with a request/order for a package containing the requested object. This package is delivered as a *Dissemination Information Package (DIP)* from the AIS. This DIP is de-constructed and PoF internal processing commences, including e.g. Contextualizer processing. When internal processing is finished, the Archiver commence with packaging the object(s) and relevant metadata into a structure described using CMIS. The receiving system is notified on the existence of the prepared delivery, and fetches it from the PoF. As in the Ingest workflow, the local working copies are removed after a successful delivery of the objects to the receiving system.

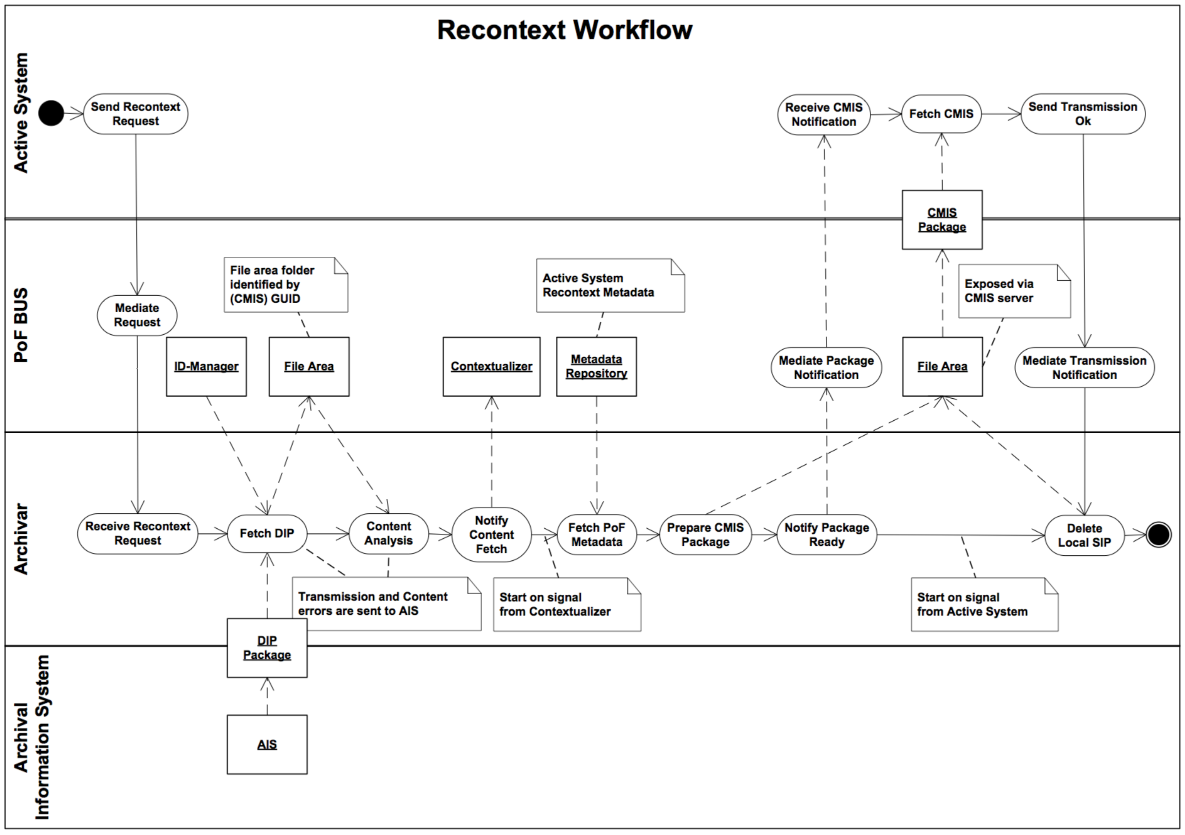

Figure 6: Recontextualization workflow for access

3.3.1 Communication

Communication will chiefly be with PoF Adapters to the active systems, and with the Preservation system. However all components in the PoF middleware that prepares the data objects in some way, are also candidates for communication. Most, or all, of this would be handled by the PoF Bus.

3.4 Preserve-or-Forget Internal Processing

Essentially, the workflow for internal processing is a combination of the first half of the recontextualization workflow, and the second half of the Ingest workflow. This workflow has not been prioritised during the first year of the project but will be developed further together with the rest of the workflows in upcoming deliverables.

3.4.1 Communication

Apart from the communication with the AIS, all components in the PoF middleware that prepares the data objects in some way, or needs information packages from the preservation system, are candidates for communication. Most, or all, of this would be handled by the PoF Bus.

4 Prototype Implementation Specification

In this section we describe details and considerations related to the initial implementation of the Collector/Archiver component, and also some brief concepts for the Context-Aware Preservation Manager which will be developed later in the project. The Collector/Archiver component is implemented to such an extent that it is able to fetch objects from the active systems using CMIS, store them internally in the middleware for further processing by other components, and then prepare a Submission Information Package ready to be delivered to, or fetched by, the Archival Information System. For Year1 the focus has been on getting the process up and running, and the component is not extensively tested with large workloads, but is able to handle the type of objects and packages that we are using at this stage.

4.1 Archiver

4.1.1 Description

Triggered by a request from the active system(s), this component collects data objects that should be preserved and prepares and submits them to the preservation system by packaging the data objects together with relevant metadata (into a SIP). The component receives a preservation request in the form of either a REST-AtomPub reference or a JSON request (in CMIS called "Browser Binding"). Acting on that request, the Collector fetches the object and metadata (CMIS) from the reference provided. The Collector notifies the ID-manager with the CMIS-Id (GUID) of the object, and notifies the PoF Bus that an object has been collected. Before a Submission Information Package (SIP) can be created, other PoF components need to process the object and extract relevant metadata and other characteristics needed for the forgetting process. This metadata is stored in the Metadata Repository, and on a trigger from the bus, the Archiver fetches metadata and prepares the package and submits it to the Preservation System. There are at least two options here: 1. The Archiver sends a reference to where the Preservation System can fetch the package; 2. The Archiver sends the package to the ingest folder of the Preservation System. In response to the submission, the Archiver needs an archive ID that should be sent to the ID-manager. The Collector/Archiver is also responsible for restructuring DIPs into packages that the active system can handle to get the information back into active use. As a response to a trigger, that can come from the active system, or from PoF internal components (e.g. the scheduler), a request is made to the Preservation System for a DIP. The DIP is then disassembled and restructured if needed for adoption in the active system. This may include restructuring of metadata in order to facilitate ingest into the active system. Transformation of content objects is not considered to be a part of this functional entity.

4.1.2 Details

The Collector/Archiver is active in both ingest and access workflows and provides a common interface to the active systems (i.e. information systems at the Producer side of the OAIS model). It exchanges objects with the active systems in the form of CMIS objects.

The software is implemented on a Java platform using MySQL as relational database. Code documentation excerpts can be found in appendix A and appendix B. The system is tested on Ubuntu (Linux) platform. Software API packages used:

- Java
- MySQL 5.0
- METS API 1.0
- DROID 4
- Md5deep
- Apache Commons Compress API

The database is used for storing the data needed for construction of the METS metadata file. The METS is created using a modified version of the METS API 1.0 to accommodate MODS as well as XML headers. The DROID software is used to extract the MIME type from information objects and will later on be used for more extensive analysis. Checksums are generated using the Md5deep component. At the moment we support MD5, SHA1 and SHA256. The Apache Commons Compress API is used to compress/package the objects to either a zip-file or a tar-package.

A sequence diagram describing the internal processing of the Archiver component extracting metadata relevant for creation of a SIP can be seen in Figure 7. Figure 8 shows the internal processing sequence for preparation of a SIP.

For the CMIS communication we utilize a combination of Java code and Groovy scripts in order to fetch objects from the active systems. For exposing objects to the active systems, there are currently two approaches that are evaluated. One possible approach is to use the OpenCMIS FileShare Repository[2]. It does come with a “not intended for production use” warning, but it would serve the purpose of demonstration for Y1. The project opted for the other alternaltive, which is to use a more production scale solution, in this case Alfresco[3] so that we already have it in place for the coming years.

[2] `http://chemistry.apache.org/opencmis-fileshare-repository.html`
[3] `http://www.alfresco.com/cmis`

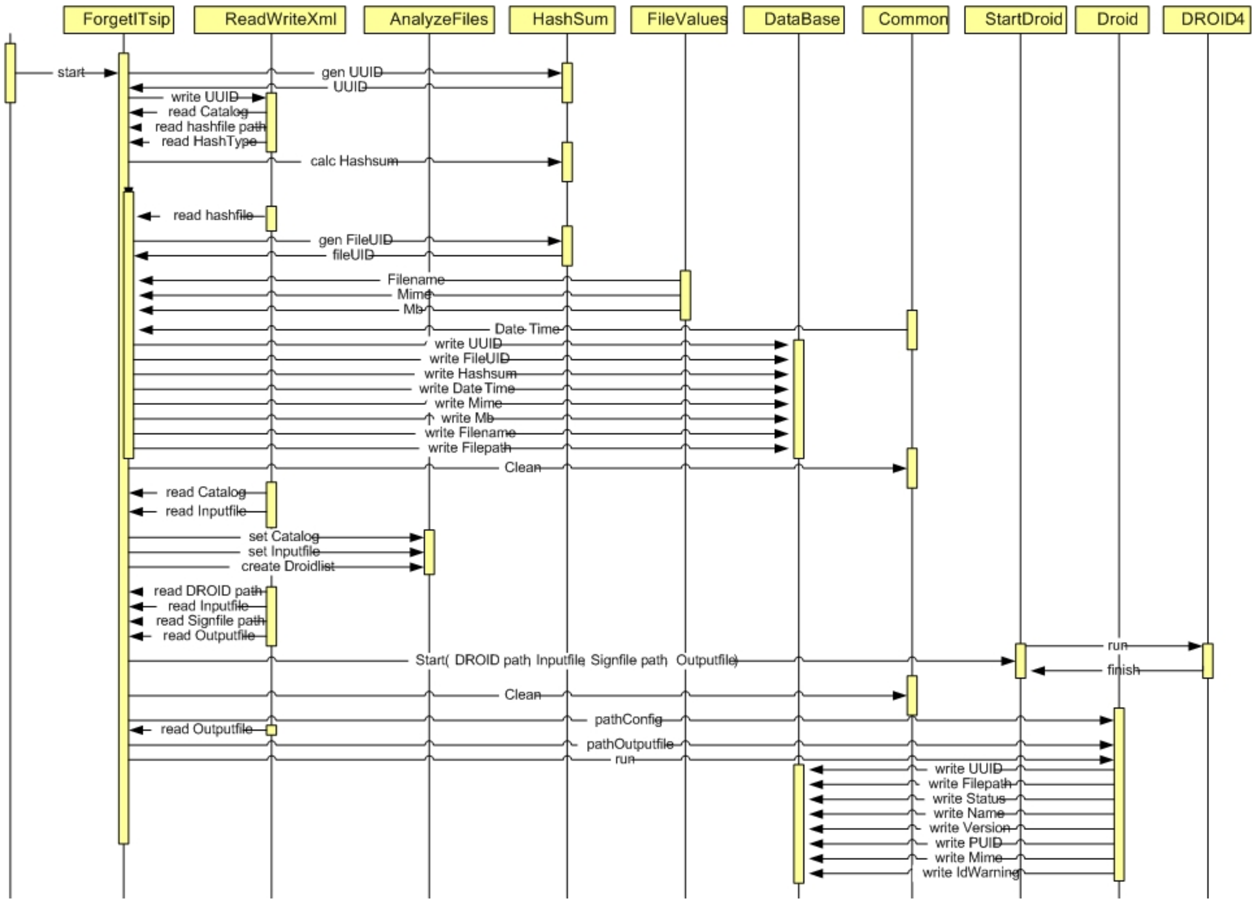

Figure 7: Sequence Diagram for Extracting Archiver Metadata

4.2 Context-Aware Preservation Manager

4.2.1 Description

The purpose with the Context-Aware Preservation Manager is to partly serve as an helping hand for the function of the Preservation Planning entity, and to some extent the Administration entity, in the AIS (Archive Information System / Preservation System). Those functions need to be stretched out to meet the active systems (and their owners) and this is where the Context-Aware Preservation Manager comes in. There exists a need to handle changes on both sides of the middleware, which includes enabling communication of events and triggers relevant for both the preservation systems and the (owners of the) active systems. As an example, the preservation systems have internal preservation plans which might include transformation of objects at ingest, if the objects are in unsuitable formats. These “default transformations” need to be communicated to the active systems. This may be communicated already at (or before) ingest, since these plans are known beforehand. As another example, the AIS is responsible for preservation of the objects for long term, but the ForgetIT system must be able to re-contextualize the objects into

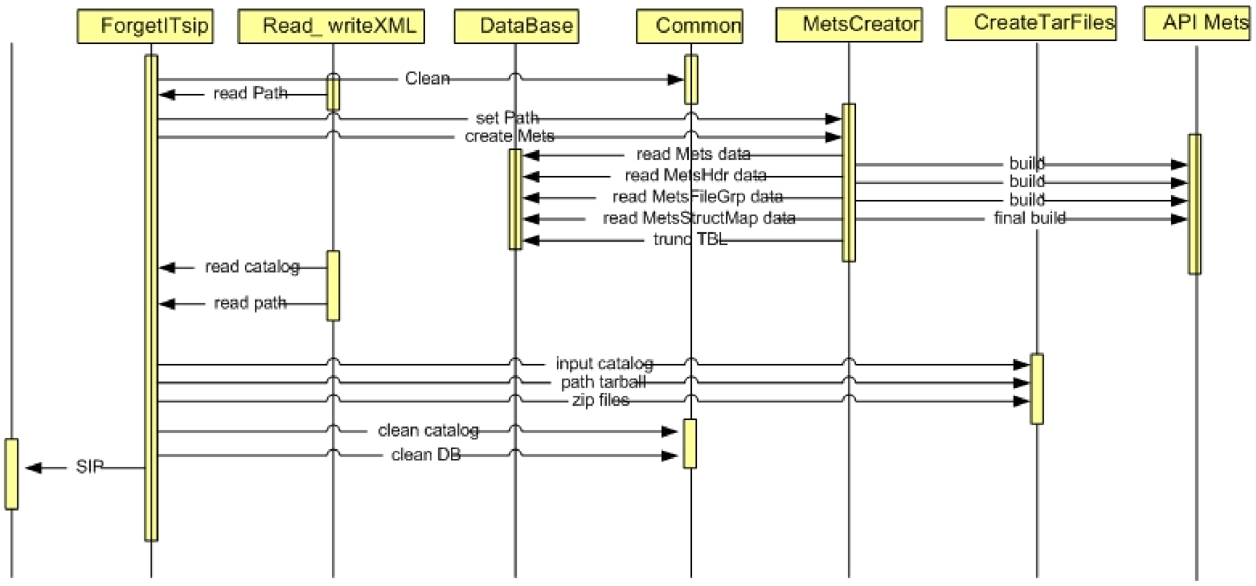

Figure 8: Sequence Diagram for Creating a SIP

active systems. This means that when the AIS makes a decision to transform a class of objects - this must be communicated to the active system and its owners. If this transformation would ruin the chances of re-contextualization, some actions need to be taken to ensure the possibility of re-contextualization (e.g. transformation to another format for re-contextualization).

4.3 Middleware solution: the Message Oriented Middleware (MOM)

The interdependency between middleware components, and the processing time that might be required in some instances, means that we have to queue events so that they are handled in the correct order, especially since some processing may not start before certain other processes have finished. There exists several options on how to achieve this, but this section focuses on the approach chosen for the project. The detailed descriptions will be available in deliverables from WP8, but introductory description to the chosen approach is discussed in the next section and will be refined in future releases of the document.

The field of middleware solutions and Service Oriented Architecture (SOA) includes several solutions and approaches which cannot be discussed here. We will just mention here that the integration of distributed applications to build complex workflows requires service orchestration and choreography, in order to implement business logic. Commercial and open source solutions are available for both platforms and workflow managers. Such solutions still play a key role in the IT market and all major vendors provide solutions for building enterprise systems. Among the commercial solutions it is worth mentioning Microsoft BizTalk, XMLBus by Iona, WebSphere by IBM or Oracle SOA Suite. The open source market includes several mature solutions such as OpenFlow, Taverna (which recently gained popularity in the digital preservation field), Mule, Apache Service Mix and several others. Many open source systems offer for free a high level of stability and several features, which can fulfil the requirements of almost all users. In the following a few details will be provided about Apache solutions, because they are considered a good choice for ForgetIT objectives.

The solutions mentioned above are all built on top of a middleware based on messaging, which became a popular approach in the last decade, when the concepts of Enterprise Service Bus and Message Oriented Middleware were formalized. A good introduction and still a reference manual for all interested developers is the book from David A. Chappel about ESB and MOM [6]. Many concepts and figures reported in the following have been borrowed from that guide.

The top open source solutions, Mule and Apache ServiceMix, are built on top of a MOM. In the following we describe briefly the main concepts related to MOM since this approach will be adopted in ForgetIT for implementing the PoF middleware.

A Message Oriented Middleware (MOM) is a key part of an ESB architecture and is a concept that involves data transfer between applications using a communication channel carrying messages, i.e. self-contained units of information. In a MOM-based communication environment, messages are usually sent and received asynchronously.

One of the main advantages of using a MOM is that message-based communications enable decoupling of distributed applications, since senders and receivers of messages are never aware of each other. The messages are sent to and received from a messaging system, which is in charge for delivering each message to the appropriate destination.

Using message-based communications, applications are abstractly decoupled; senders and receivers are never aware of each other. Instead, they send and receive messages to and from the messaging system. It is the responsibility of the messaging system (MOM) to get the messages to their intended destinations. In a messaging system, an application uses an API to communicate through a messaging client that is provided by the MOM vendor. The messaging client sends and receives messages through a messaging system, as shown in Figure 9.

The messaging system is responsible for managing the connection points between multiple messaging clients, and for managing multiple channels of communication between the connection points. The messaging system is usually implemented as a software process, which is commonly known as a message server or a message broker. Message servers are usually capable of being grouped together to form clusters that provide advanced capabilities such as load balancing, fault tolerance, and sophisticated routing using managed security domains.

Messaging enables a loosely coupled environment in which an application does not need to know the intimate details of how to reach and interface with other applications. In choosing a type of communication infrastructure, it is important to consider the trade-offs between loosely coupled and tightly coupled interfaces, and asynchronous and synchronous

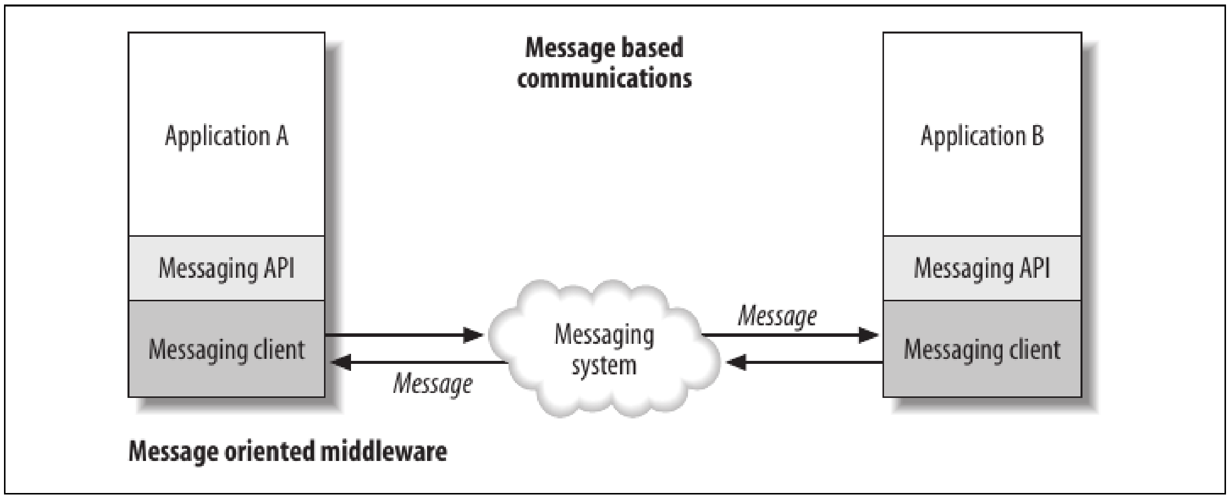

Figure 9: Communication in a MOM. Picture taken from [6]

interaction modes. This feature distinguishes messaging from RPC-style programming, as discussed above, which is typically synchronous. When performing a synchronous operation across multiple processes, the success of one RPC call depends on the success of all the downstream RPC calls that are part of the same synchronous request/response cycle. This makes the whole invocation an all-or-nothing proposition and additional control software and logic must be provided to handle failures. One of the driving ideas behind the concept of bus is the combination of loosely coupled interfaces and asynchronous interactions, in analogy with hardware bus architecture. As explained in [6], ”*Everything on the bus is accessible by anything else on the bus. If you are the owner of a service or an application domain, you need only be concerned with three operations: plugging into the bus, posting data to the bus, and receiving data from the bus. The bus then gets the data to the applications in the target data formats*”. Having this in mind, it can be easily understood how the MOM approach, based on standard APIs for communication and transport, can be considered a valuable solution for implementing a bus in an enterprise environment.

Two different messaging models can be used, depending on the requirements of the application: one makes use of topics and the other of queues. In the case of topics, the Producer and the Consumer are referred to as Publisher and Subscriber, respectively. This model can be used for a one-to-many broadcast of information: multiple consumers may register an interest with, or subscribe to a topic. A producer sends a message on that channel by publishing on that topic. Each subscriber receives a copy of that message. In the case of queues, the Producer and the Consumer are referred to as Sender and Receiver, respectively. This model can be used for a one-to-one communication between two specific applications. Only one consumer may receive a message that is sent to a queue. Figure 10 below depicts the two models. The choice of which model to use can largely depend on how many consumers need to see duplicate copies of the same message.

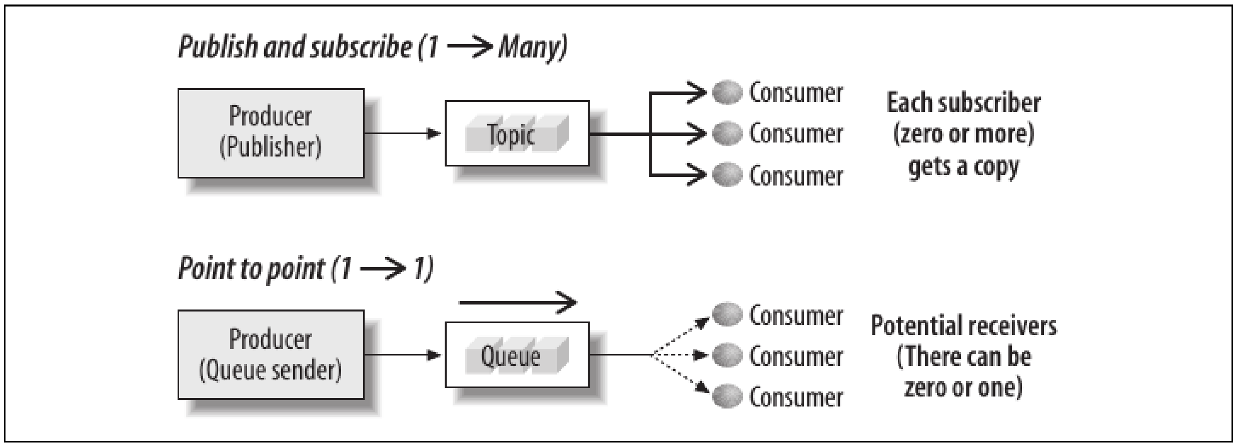

Figure 10: MOM messaging model with topics (Publish and Subscribe) and queues (Point to Point). Picture taken from [6]

A point-to-point queue may have multiple consumers typically listening for the purposes of load-balancing: only one receiver may consume each individual message. There may also be no receivers listening, in which case the message stays in the queue until a receiver is registered on the message server for that queue. In the publish-and-subscribe model, if no subscribers are registered for a given topic, messages could be discarded. This behaviour can be further changed by changing configuration for reliability and persistence.

A message is typically composed of three basic parts: the headers, the properties, and the message payload or body (Figure 11). The headers are used by both the messaging system and the application developer to provide information about things such as the destination, the reply-to destination, the message type, and the message expiration time.

XML is the favourite technology for representing data as it flows between applications across the ESB. The data that is produced and consumed by a vast array of applications can exist in a variety of formats and packaging schemes. While it is certainly possible for the ESB to carry data using any form of packaging or enveloping scheme, representing in-flight data as XML provides several benefits, including the ability to use specialized ESB services that combine data from different sources to create new views of data, and to enrich and re-target messages for advanced data sharing between applications. Therefore, one of the benefits of using XML as the native data format for the ESB is that messages are not treated as opaque chunks of data. If all data between applications and services is formatted as XML documents, underpinnings are provided by the ESB that allow you to layer advanced capabilities on top of the ESB to gain real-time insight into the business data that flows through the enterprise.

Several standards are available for message transmission. The most popular one or at least the one which is supported by all major MOM implementations is JMS. Even if JMS has its roots in Java, clients compliant to JMS written in different languages are

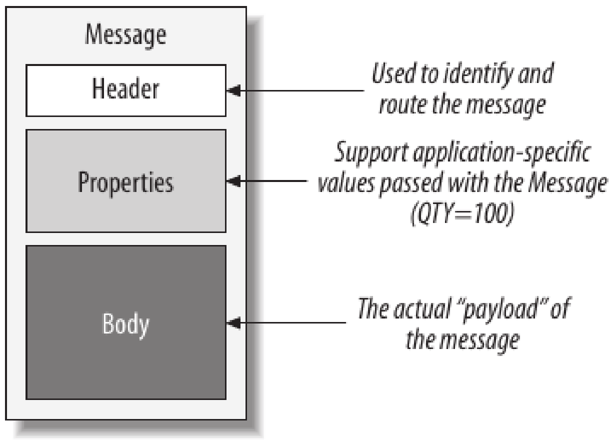
Message
Header
Properties
Body
Used to identify and route the message
Support application-specific values passed with the Message (QTY=100)
The actual "payload" of the message

Figure 11: Structure of messages in a MOM (the header is used for correct routing between applications and the body transports the actual data to be processed). Picture taken from [6]

available, too. JMS supports a set of messaging patterns and helper interfaces to synchronously and asynchronously support the request/reply pattern using both the publication/subscription and point-to-point messaging models and is the reference technology for standards-based integration, as already discussed above.

An enterprise MOM can be used at the core of the ESB communication layer. The MOM can be considered as a multi-protocol messaging bus that supports asynchronous delivery of messages.

4.3.1 Chosen Approach

The Preserve-or-Forget architecture, which has already been described in D8.1 [7], includes four layers corresponding to user applications, the PoF middleware, the archive and the cloud storage. The workflow model described in this document is mainly relevant for the PoF middleware implementation, as already mentioned above. The architecture diagrams reported in D8.1 include also the concept of middleware bus as a way to enable communication and transport among all integrated components.

In D8.1 two simple integrated workflows have been described, one focusing on a basic synergetic preservation, and the other workflow is for managed forgetting. For the two workflows we provide simple activity diagrams discussing the architecture components involved.

The proposed approach was to implement a simple Message Oriented Middleware in order to reduce the complexity of the integration using a lightweight standard approach. The concept of middleware bus is implemented by the message broker. Using a MOM with a JMS broker to exchange data between applications in the form of XML messages should be enough for the first release of the PoF framework, where the focus will be on the development of each component and on their early integration to demonstrate the aforementioned workflows in D8.1. However, since MOM solutions are the backbone for almost any enterprise-class middleware, this approach does not prevent the adoption of more complex solutions for implementing the middleware. The bottom line of D8.1 was to choose a standard open source implementation of a message broker which should support additional features for simple orchestration of the processes.

Currently several solutions are available on the market or from the open source community to implement a MOM. They choice can be made according to different criteria. For the scope of ForgetIT, an open source solution, which is also mature and supported by an active community, seems to be the best candidate. The implementation should be based on standard technologies and protocols, to avoid vendor lock-in for the future exploitation of the project results. An other criterion is the availability of detailed documentation and support.

Some possible candidates have already been mentioned in D8.1. Among them, the different components of the Apache ServiceMix [8] suite, specifically ActiveMQ [9] and Camel [10], look quite promising. The former is probably the most widely adopted implementation of a message broker based on Java specification (JMS). ActiveMQ is included in several other solutions such as Mule. The ServiceMix suite includes also the Camel component, which can be used for process orchestration and integrates seamlessly with ActiveMQ.

ActiveMQ provides several functionalities which have been discussed above for message servers, including support for topics and queues, reliability and persistence, independence from the specific language.

Apache Camel is a lightweight integration framework which implements all EIPs, enabling easy integration of different applications using the required patterns. Camel supports different languages and technologies, including Java, Spring XML, Scala or Groovy, HTTP, FTP, JMS, EJB, JPA, RMI, JMX, LDAP and others. Besides, own custom components can be created. Camel can be deployed as standalone application, in a web container (e.g. Tomcat or Jetty), in a JEE application Server (e.g. JBoss AS or WebSphere AS), in an OSGi environment or in combination with a Spring container. The approach used in Camel for application integration is independent of the particular domain specific language or technology.

Both ActiveMQ and Camel are mature and production ready and their combination offers also scalability, transaction support, concurrency and monitoring.

The approach discussed above will be detailed in D8.3 [D8.3], expected at M18, which will describe the first release of the PoF Framework and will include details about the actual implementation of the middleware.

5 Conclusions and Future Work

This deliverable builds on work from D5.1 with more focus on implementation specifics. The prototype component (Archiver) was discussed and implemented. The most interesting part still remains, when we will integrate the component, together with components from other workpackages, into the ForgetIT framework. This will also give us indications on what needs to be further developed in the Archiver, as well as input to the development of the Context-Aware Preservation Manager. Consideration will be put on that although the Archiver is a component that mainly works “in the dark”, the Context-Aware Preservation Manager most likely will have more user interaction, and thereby also require more user evaluation.

In the second year of the project, the Archiver component will be further enhanced with support for file format identification on version level, while also spending time on rectifying problems that might show up during integration in the ForgetIT framework. Work will also commence on the Context-Aware Preservation Manager.

References

[1] Joachim Korb and Stephan Strodl. Digital preservation for enterprise content: A gap-analysis between ecm and oais. In *Proceedings of the 7th International Conference on Preservation of Digital Objects*, pages 221–229. Österreichische Computer Gesellschaft, 2010. Vortrag: IPRES 2010, Wien; 2010-09-19 – 2010-09-24.

[2] Seamus Ross. Digital preservation, archival science and methodological foundations for digital libraries. *New Review of Information Networking*, 17(1):43–68, 2012.

[3] Producer-archive interface methodology abstract standard. Recommendation for standard, Consultative Committee for Space Data Systems, Office of Space Communication (Code M-3), NASA, Washington, DC 20546, USA, May 2004.

[4] Reference model for an open archival information system. magenta book. issue 2. june 2012. Technical report, Consultative Committee for Space Data Systems, CCSDS Secretariat, Space Communications and Navigation Office, 7L70 Space Operations Mission Directorate, NASA Headquarters Washington, DC 20546-0001, USA, 2012. CCSDS 650.0-M-2.

[5] Christian Keitel. Ways to deal with complexity. In *Proceedings of The Fifth International Conference on Preservation of Digital Objects Joined Up and Working: Tools and Methods for Digital Preservation*, pages 287–291, 2008.

[6] David A. Chappell. *Enterprise Service Bus*. O'Reilly, 2004.

[7] ForgetIT. D8.1 - Integration Plan and Architectural Approach. `http://www.forgetit-project.eu/fileadmin/fm-dam/deliverables/ForgetIT_WP8_D8.1.pdf`, November 2013. Retrieved: 31 January 2014.

[8] Apache ServiceMix. `http://servicemix.apache.org`. Retrieved: 31 January 2014.

[9] Apache ActiveMQ. `http://activemq.apache.org`. Retrieved: 31 January 2014.

[10] Apache Camel. `http://camel.apache.org`. Retrieved: 31 January 2014.

A Package forgetitsip

Package Contents *Page*

Class AnalyzeFiles

Declaration

public class AnalyzeFiles
extends java.lang.Object

Fields

- protected static java.lang.String **L_pathKatalog**
- protected static java.lang.String **L_droidLista**

Constructors

- **AnalyzeFiles**
  `public` **AnalyzeFiles**`()`

Methods

- **createFileList**
  `public static boolean` **createFileList**`()`
  - **Returns** – boolean true/false
  - **See also**
    - Method that creates a file list containing the path to the files.
- **createFileListForDroid**
  `public boolean` **createFileListForDroid**`()`
  - **Returns** – boolean true/false
  - **See also**
    - Creates a file list of the files that are unpacked in a directory, file list used by DROID4.
- **setDroidList**
  `public void` **setDroidList**`(java.lang.String` **droidLista**`)`
  - **Parameters**
    - `droidLista,` – output for droid list incl. filename.
- **setPathCatalog**
  `public void` **setPathCatalog**`(java.lang.String` **pathkatalog**`)`
  - **Parameters**
    - `pathkatalog,` – Pointing to the directory containing the folder content, metadata and systems.

Class Common

See also

– A class that contains common methods

Declaration

public class Common
extends java.lang.Object

Constructors

- **Common**
  `public` **Common**`()`

Methods

- **dateAndTime**
  `public java.lang.String` **dateAndTime**`()`
  - **Returns** – date and time stamp.
  - **See also**
    * Creates a date and time stamp.
- **delAllFilesCatalog**
  `public boolean` **delAllFilesCatalog**`(java.lang.String` **path**`)`
  - **Returns** – boolean true/false
  - **See also**
    * A general method to delete files in a directory.
- **deleteFile**
  `public boolean` **deleteFile**`(java.lang.String` **pathtofile**`)`
  - **Parameters**
    * `pathtofile,` – path including the file name.
  - **Returns** – boolean true/false
  - **See also**
    * Deletes the given file
- **generateUID**
  `public java.lang.String` **generateUID**`()`
  - **Returns** – unique UUID
  - **See also**
    * Creates a unique UUID.

Class CreateTarFile

See also

– A class that creates tar or zip file.

Declaration

public class CreateTarFile
extends java.lang.Object

Constructors

- **CreateTarFile**
  `public` **CreateTarFile**`()`

Methods

- **setInputFileCatalog**
  `public void` **setInputFileCatalog**`(java.lang.String` **inputFileCatalog**`)`
- **setPathTarBall**
  `public void` **setPathTarBall**`(java.lang.String` **pathTarBall**`,`
  `java.lang.String` **tarFileName**`)`
  - **Parameters**
    * `pathTarBall`, – path to the directory where the tar, zip file to be created.
    * `tarFileName`, – filename of the tar or zip file.
- **tarFiles**
  `public void` **tarFiles**`(java.lang.String` **compression**`)`
  - **Parameters**
    * `compression`, – compression method (tar—zip).
  - **See also**
    * compresses the file to tar or zip.

Class DataBase

See also

– A class that writes and retrieves information from the MySQL DB.

Declaration

public class DataBase
extends java.lang.Object

Fields

- protected java.sql.ResultSet **rSet**

Constructors

- **DataBase**
  `public` **DataBase**`()`

Methods

- **getDataForMets**
  `public java.sql.ResultSet` **getDataForMets**`(java.lang.Integer` **selectSQL**`) throws java.lang.Exception`
  - **Parameters**
    * `selectSQL`, – selector for SQL string (1-5).
  - **Returns** – ResultSet
  - **See also**
    * A method that retrieves data from MySQL database.
- **getNrOfFiles**
  `public java.lang.Integer` **getNrOfFiles**`()`
  - **Returns** – Return the number of records in the table..
  - **See also**
    * One method counts the number of records in the table (the Mets).
- **readUidToTxt**
  `public java.lang.String` **readUidToTxt**`()`

  - **Returns** – Return unique identifiers from the database.

- **setCloseDbForRs**
  `public void` **setCloseDbForRs**`() throws java.lang.Exception`
  - **See also**
    - A method closes the database, is used for RS
- **setDbForRs**
  `public boolean` **setDbForRs**`()`
  - **Returns** – boolean true/false
  - **See also**
    - A method of creating a database connection, used for RS.
- **setPathConfig**
  `public void` **setPathConfig**`(java.lang.String` **pathConfig**`)`
- **truncTable**
  `public boolean` **truncTable**`(java.lang.String` **tblName**`)`
  - **Returns** – boolean true/false
  - **See also**
    - A method that truncates table in mySQL DB.
- **writeToDroidInfoDB**
  `public boolean` **writeToDroidInfoDB**`(java.lang.String` **paketUID**`, java.lang.String` **filePath**`, java.lang.String` **Status**`, java.lang.String` **Name**`, java.lang.String` **Version**`, java.lang.String` **PUID**`, java.lang.String` **mimeValue**`, java.lang.String` **idWarning**`)`
  - **Returns** – boolean true/false
  - **See also**
    - A method writes the data from DROID to sip_db DB (mySQL).
- **writeToMetsDB**
  `public boolean` **writeToMetsDB**`(java.lang.String` **paketUid**`, java.lang.String` **filUid**`, java.lang.String` **filNamn**`, java.lang.String` **filDatum**`, java.lang.String` **Mime**`, java.lang.String` **Version**`, java.lang.String` **Byte**`, java.lang.String` **hashSum**`, java.lang.String` **hashsumType**`, java.lang.String` **filePath**`)`
  - **Returns** – boolean true/false
  - **See also**
    - A method writes the data to (tbl: mets) to the sip_db DB (mySQL).

- **writeToPaketinfoDB**
  `public boolean` **writeToPaketinfoDB**`(java.lang.String` **paketBesk**`, java.lang.String` **paketDatum**`, java.lang.String` **levOrg**`, java.lang.String` **kontaktNamn**`, java.lang.String` **kontaktTele**`, java.lang.String` **kontaktMail**`, java.lang.String` **sipMjukvara**`, java.lang.String` **arkivSkaparNamn**`, java.lang.String` **arkivSkaparOrgnr**`, java.lang.String` **levSysNamn**`, java.lang.String` **bevOrgNamn**`, java.lang.String` **bevOrgID**`, java.lang.String` **levTyp**`, java.lang.String` **levSpec**`, java.lang.String` **levDatum**`, java.lang.String` **levOverKommelse**`, java.lang.String` **dokumentId**`, java.lang.String` **paketUID**`)`
  - **Returns** – boolean true/false
  - **See also**
    - A method writes the data to (tbl: mets) to the sip_db DB (mySQL).

Class Droid

See also

– A class that takes information out of the Droid output and sends it to the table in the DB.

Declaration

public class Droid
extends java.lang.Object

Constructors

- **Droid**
  `public` **Droid**`()`

Methods

- **droidOutputXmlToDB**
  `public void` **droidOutputXmlToDB**`()`
  - **See also**
    - Reads the information from xml file and make calls to the method that writes the information to the DB.

- **setDroidOutputXml**
  `public void` **setDroidOutputXml**`(java.lang.String` **droidOutputXml**`)`
  - **Parameters**
    - `droidOutputXml` – path to output file from Droid (even filename).
  - **See also**
    - Sets the path on the Droid output file.

- **setPathConfig**
  `public void` **setPathConfig**`(java.lang.String` **pathConfig**`)`
  - **Parameters**
    - `pathConfig,` – path to config file.
  - **See also**
    - Sets the path to the config file.

Class FileValues

See also

– A class that produces values and properties of files.

Declaration

public class FileValues
extends java.lang.Object

Constructors

- **FileValues**
  `public` **FileValues**`()`

Methods

- **createMeFiles**
  `public boolean` **createMeFiles**`(java.lang.String` **pathtocatalog**`)`
  - **Parameters**
    - `pathtocatalog,` – path to catalog where files are.
  - **Returns** – boolean, true/false

- **See also**
  - Method that sets the values for the files to be written in ADDML structure

- **fileData**
  `public boolean` **fileData**`(java.lang.String` **path**`)`
  - **Parameters**
    - `path,` – path to file.
  - **Returns** – boolean, true/false
  - **See also**
    - Method that sets the values for the input file, name, mime, mb,AD

- **getFilename**
  `public java.lang.String` **getFilename**`()`
  - **Returns** – filename

- **getFileversion**
  `public java.lang.String` **getFileversion**`()`
  - **Returns** – file version
  - **See also**
    - Used only for Web.

- **getMB**
  `public java.lang.String` **getMB**`()`
  - **Returns** – return files size in mb.

- **getMimevalue**
  `public java.lang.String` **getMimevalue**`()`
  - **Returns** – mime value.

Class ForgetItSip

Declaration

public class ForgetItSip
extends java.lang.Object

Fields

- protected static java.lang.String **hashsumType**
- protected java.lang.String **filepath**

- protected static java.lang.String **pathConfig**
- protected static java.lang.String **dbFullpath**
- protected static java.lang.String **dbXlinkPath**
- protected static java.lang.String **dbChecksum**
- protected static java.lang.String **dbMime**
- protected static java.lang.String **dbSize**
- protected static java.lang.String **dbUse**
- protected static java.lang.String **dbDate**
- protected static java.lang.String **dbFileID**
- protected static java.lang.String **dbPaketuid**

Constructors

- **ForgetItSip**
  `public` **ForgetItSip** `()`

Methods

- **main**
  `public static void` **main** `(java.lang.String[]` **args**`)`
  - **See also**
    - main,method that runs the other classes.

Class HashSum

See also

- A class that creates the hash sums for files in a directory and
- generates a text file with the path and the hash sum.

Declaration

public class HashSum
extends java.lang.Object

Constructors

- **HashSum**
  `public` **HashSum**`(java.lang.String` **katalog**`,`
  `java.lang.String` **sokvag_filnamn**`)`
  - **Parameters**
    - `katalog` – , path to file folder.
    - `sokvag_filnamn` – , path and name of output txt file for hashsum.
  - **See also**
    - constructor for the class.

Methods

- **createHashsum**
  `public boolean` **createHashsum**`()`
  - **Returns** – boolean true/false
  - **See also**
    - A method that creates a MD5 or SHA1 list of files given cataloged.
- **createHashsumforfile**
  `public boolean` **createHashsumforfile**`(java.lang.String` **fullsokvag**`)`
  - **Parameters**
    - `fullsokvag,` – absolute path to the file.
  - **See also**
    - A method that creates the hash sum for one file.
- **generateFileUID**
  `public java.lang.String` **generateFileUID**`()`
  - **Returns** – file UUID
  - **See also**
    - Creates a unique identifier UUID for a file.
- **generateUID**
  `public java.lang.String` **generateUID**`()`
  - **Returns** – UUID
  - **See also**
    - Creates a unique identifier UUID.
- **setAlgoritm**
  `public java.lang.String` **setAlgoritm**`(java.lang.String` **valAlgoritm**`)`

  - **Parameters**
    - `valAlgoritm` – set md5 or sha1.
  - **See also**
    - Provides an opportunity to choose between MD5 — SHA1.

- **setCatalog**
  `public void` **setCatalog**`(java.lang.String` **sKatalog**`)`
  - **Parameters**
    - `sKatalog,` – path to folder.
  - **See also**
    - path to file folder.

- **setOutFile**
  `public void` **setOutFile**`(java.lang.String` **sUtfil**`)`
  - **Parameters**
    - `sUtfil,` – path and name of output txt file for hashsum.
  - **See also**
    - set the output txt file for hash sum.

Class MetsCreator

See also

– A class that creates a mets file for the tar—zip package.

Declaration

public class MetsCreator
extends java.lang.Object

Constructors

- **MetsCreator**
  `public` **MetsCreator**`()`

Methods

- **createMets**
  `public void` **createMets**`() throws METSException, java.io.FileNotFoundException, org.xml.sax.SAXException, javax.xml.parsers.ParserConfigurationException, java.io.IOException, java.lang.Exception`
  - **See also**
    - A method that creates a mets file.
- **setPathConfig**
  `public void` **setPathConfig**`(java.lang.String` **pathConfig**`)`
  - **Parameters**
    - `pathToMetsfile`, – absolute path to the config file.
  - **See also**
    - Path to the config file (including file name).
- **setPathToMetsfile**
  `public void` **setPathToMetsfile**`(java.lang.String` **pathToMetsfile**`)`
  - **Parameters**
    - `pathToMetsfile`, – Absolute path to the file.
  - **See also**
    - Sets the path to the Mets file (including filename).

Class ReadWriteXml

See also

– Reads from config file and writes to the config file.

Declaration

public class ReadWriteXml
extends java.lang.Object

Constructors

- **ReadWriteXml**
  `public` **ReadWriteXml**`()`

Methods

- **readConfigfile**
  `public java.lang.String` **readConfigfile**`(`
  `java.lang.String` **pathconfigfile**`,`
  `java.lang.String` **tag**`)`
  - **Parameters**
    - `pathconfigfile,` – path to config file.
    - `Strng,` – tag name where the value is to be retrieved.
  - **Returns** – value from the given tag.
  - **See also**
    - Lser ut data ur xml configurationsfil
- **writeConfigfile**
  `public void` **writeConfigfile**`(java.lang.String` **pathconfigfile**`,`
  `java.lang.String` **tag**`, java.lang.String` **value**`)`
  - **Parameters**
    - `pathconfigfile` – , path to config file.
    - `tag` – , tag name into which the value should be written.
    - `value` – , value to be written down.
  - **See also**
    - Method that writing data to XML configurations file.
- **writeToXmlTagOrAttribByIdx**
  `public boolean` **writeToXmlTagOrAttribByIdx**`(`
  `java.lang.String` **pathofxmlfile**`, java.lang.String` **tag**`,`
  `java.lang.String` **attr**`, java.lang.String` **value**`, int` **nodeIdx**`)`
  - **Parameters**
    - `pathofxmlfile` – , path to config file.
    - `tag,` – tag name.
    - `attr,` – attribute.
    - `value,` – the value to be written to the attribute.
    - `nodeIdx,` – which node in the ordering to be written to.
  - **Returns** – boolean, true/false.
  - **See also**
    - Writes data to XML configurations file, tags with attributes

Class StartDroid

See also

– A class that starts Droid, fixes issues with Boot (bug problem).

Declaration

public class StartDroid
extends java.lang.Object

Constructors

- **StartDroid**
  `public` **StartDroid** `()`

Methods

- **main**
  `public static void` **main** `(java.lang.String` **pathDroid**`,`
  `java.lang.String` **argA**`, java.lang.String` **argS**`,`
  `java.lang.String` **argO**`)`
  - **See also**
    - A main method that starts Droid.

B Package eu.forgetit.ltu.mavencmisclient

Package Contents *Page*

Class CMISapp

Declaration

public class CMISapp
extends java.lang.Object

Fields

- protected static java.lang.String **pathConfig**
- protected static java.lang.String **workPath**
- protected static java.lang.String **cmisIdFolder**

Constructors

- **CMISapp**
  `public` **CMISapp**`()`

Methods

- **main**
  `public static void` **main**`(java.lang.String[]` **args**`)`
  `throws java.io.IOException, java.lang.InterruptedException`

- **See also**
  - main method for the class

Class CopyDirFile

See also

– copy one file or a entire folder with files

Declaration

public class CopyDirFile
extends java.nio.file.SimpleFileVisitor

Constructors

- **CopyDirFile**
  `public` **CopyDirFile**`(java.nio.file.Path` **fromPath**`,`
  `java.nio.file.Path` **toPath**`,`
  `java.nio.file.CopyOption` **copyOption**`)`

Methods

- **copyFile**
  `public void` **copyFile**`(java.io.File` **sFile**`, java.io.File` **dFile**`)`
  `throws java.io.IOException`
  - **Parameters**
    - `sFile` – srcPath, filepath and filename
    - `dFile` – destPath, filepath and filename
  - **Returns** –
  - **See also**
    - copy one file from folder1 to folder2
- **preVisitDirectory**
  `public java.nio.file.FileVisitResult` **preVisitDirectory**`(`
  `java.nio.file.Path` **dir**`,`
  `java.nio.file.attribute.BasicFileAttributes` **attrs**`)`
  `throws java.io.IOException`

- **visitFile**
  `public java.nio.file.FileVisitResult` **visitFile**`(`
  `java.nio.file.Path` **file**`,`
  `java.nio.file.attribute.BasicFileAttributes` **attrs**`)`
  `throws java.io.IOException`

Members inherited from class SimpleFileVisitor

`java.nio.file.SimpleFileVisitor`

postVisitDirectory, preVisitDirectory, visitFile, visitFileFailed

Class ReadWriteXml

Declaration

public class ReadWriteXml
extends java.lang.Object

Constructors

- **ReadWriteXml**
  `public` **ReadWriteXml**`()`

Methods

- **readConfigfile**
  `public java.lang.String` **readConfigfile**`(`
  `java.lang.String` **pathconfigfile**`, java.lang.String` **tag**`)`
  - **Parameters**
    * `pathconfigfile`, – path to config file.
    * `tag`, – tag name where data is available.
  - **Returns** – the value of a given tag.
  - **See also**
    * Reading data out of the XML configurations file
- **writeConfigfile**
  `public void` **writeConfigfile**`(java.lang.String` **pathconfigfile**`,`
  `java.lang.String` **tag**`, java.lang.String` **value**`)`

- **Parameters**
  - `pathconfigfile,` – path to config file.
  - `tag,` – tag name where value be written.
  - `value,` – the value to be written down to config file.
- **See also**
  - Method that writes the data to xml configurations file.

Class StartPacking

See also

- calls the packing module.

Declaration

public class StartPacking
extends java.lang.Object

Constructors

- **StartPacking**
  `public` **StartPacking**`()`

Methods

- **startPacking**
  `public void` **startPacking**`(java.lang.String` **cmisUid**`)`