www.forgetit-project.eu

ForgetIT

Concise Preservation by Combining Managed Forgetting and Contextualization Remembering

Grant Agreement No. 600826

Deliverable D5.4

Work-package	WP5: Joint Information and Preservation Management
Deliverable	D5.4: Workflow model and prototype for transition between active system and AIS – Final release
Deliverable Leader	Jörgen Nilsson
Quality Assessor	Mark Greenwood
Estimation of PM spent	12
Dissemination level	PU
Delivery date in Annex I	M36
Actual delivery date	2016-03-31
Revisions	9
Status	Final
Keywords:	Preservation, ingest, re-activation, contract, manager

Revision History

Version	Major changes	Authors
0.1	Draft of structure	JN
0.2	Overall insertion of material in all sections	JN, IA
0.3	Material to section 4, 5	IA, GL
0.4	Material to section 1, 2, 3	JN, FG
0.5	Changed order of 3 & 4	JN, IA
0.6	Finalising structure	JN, IA
0.65	General check of references, figures, etc.	JN, IA
0.7	Version ready for QA	JN, IA
1.0	Added appendix, executive summary, fixed QA comments	JN, IA

List of Authors

Partner Acronym	Authors
LTU EURIX	Ingemar Andersson, Jörgen Nilsson, Göran Lindqvist, Francesco Gallo

Table of Contents

Table of figures

Acronyms

AIP Archival Information Package

AIS Archival Information System

API Application Programming Interface

AS Active System

CMIS Content Management Interoperability Services

CaPM Context-aware Preservation Manager

DIP Dissemination Information Package

DO Digital Object

DoW Description of Work

DP Digital Preservation

DPS Digital Preservation System

DROID Digital Record Object Identification

ESB Enterprise Service Bus

IS Information System

METS Metadata Encoding and Transmission Standard

MODS Metadata Object Description Standard

OAIS Open Archival Information System

OAI-PMH Open Archives Initiative – Protocol for Metadata Harvesting

PASS Preservation-Aware Storage System

PIMO Personal Information MOdel

PoF Preserve-or-Forget

REST Representational State Transfer

SD Semantic Desktop

SIP Submission Information Package

SP Storage Provider

WP Work Package

Executive summary

Increasing amounts of digital content are held by organisations and private citizens with limited know-how and awareness of digital preservation. At the same time digital assets are becoming more important in a long-term perspective (e.g. legal documents and obligations, business data, or intellectual property). Limited resources for preservation drives the need for approaches that strives for (semi-)automated preservation solutions.

An important aspect in supporting a smooth (automated) transition of digital content from source information systems to preservation systems, is hiding the complexity of the processes; providing support for decision making about alternative preservation actions, automated metadata management, quality assurance issues, handling of copies, and automated error handling.

This deliverable summarizes the final versions of the components for enabling a smooth transition between the active system and the preservation system in the ForgetIT project. The core components considered in this deliverable are the Collector/Archiver and the context-aware Preservation Manager (CaPM). The Collector/Archiver is directly involved in the transfer and exchange activities, whereas the CaPM foremost provide a Preservation Broker Contract for rule-based execution of actions.

Work described in this deliverable builds upon the approach and workflows defined in D5.1, D5.2 and D5.3, as well as on the conceptual processes defined in D8.5 for the PoF Reference Model. Although the components are in a final state in the project, there are already discussions for how to continue development of those, especially in a national (Swedish) context which also could include adopting to work produced in the E-ARK, a project co-founded by the EC (under PSP CIT).

1 Introduction

The ForgetIT project aims at helping people and organisations with decisions on what to preserve and where it should be kept, through ideas from psychology (on human memory) and with assistance of automated processes. In order to make this as transparent as possible to the users, there is a need for a smooth transition of objects between the Active Systems (the information systems that are in use by the users) and the preservation systems (the system[s] where material that should be preserved are kept). This work package (WP) describes these workflows and implements some of the functionality needed for the automated processes.

In previous WP5 deliverables, we identified a gap between content management systems and preservation systems especially regarding support for ingest of objects (D5.1) [Päivärinta et al., 2013], and we also modelled the first iteration of two workflows, one for pre-ingest and ingest, and one for re-contextualization and access (D5.2) [Nilsson et al., 2014]. D5.2 also included discussion, documentation and reasoning on the first prototype versions of the software components and the message oriented middleware approach. In D5.3 [Nilsson et al., 2015] we further elaborated on the workflows, which also included name changes influenced by the work with the reference model (D8.2) [Gallo et al., 2015a] and D8.5 [Gallo et al., 2016]. D5.3 also introduced the Context-aware Preservation Manager, although the major part of that will be seen in this report.

Tapping into these workflows is the Context-aware Preservation Manager (software) that: acts upon agreements between the users and the service providers; logs every object passing in and out of the systems; logging of process actions (activities), and any errors that might occur in the process of transferring content in and out of the systems; supports the creation and upholds submission agreements based on preservation policies, and is able to suggest actions to be taken e.g. in re-activation of material. The Context-aware Preservation Manager is also able to assist in a preservation planning scenario by providing a user interface that displays graphs with statistics based on number of objects passing through the Preserve-or-Forget (PoF) middleware per time period. This statistical data is grouped by systems involved and mime type classifications, and displays the number of objects for each mime type version. The statistical computation of object data could also be exported in a standardized format by a web service interface.

It should be noted that this report is just part of deliverable D5.4. The rest of the deliverable are the implemented components described in this report, which have been integrated into the PoF Framework.

1.1 Structure of the Report

After the Introduction, a "Big Picture" section describing circumstances related to High-level Workflows and Integration Considerations follows. The report then continues with Section 3 which describes reasoning and motivation around the Context-aware Preservation Manager, which has been the focus of year 3 of the project. Section 4 includes current workflow descriptions for Preservation preparation and Re-activation and documents what has changed since the previous release. In Section 5 the implementation details for all WP5 components, and middleware related to WP5, are described, followed by Summary and Conclusion in Section 6. Appendix A contains the schema specification for the Preservation Broker Contract.

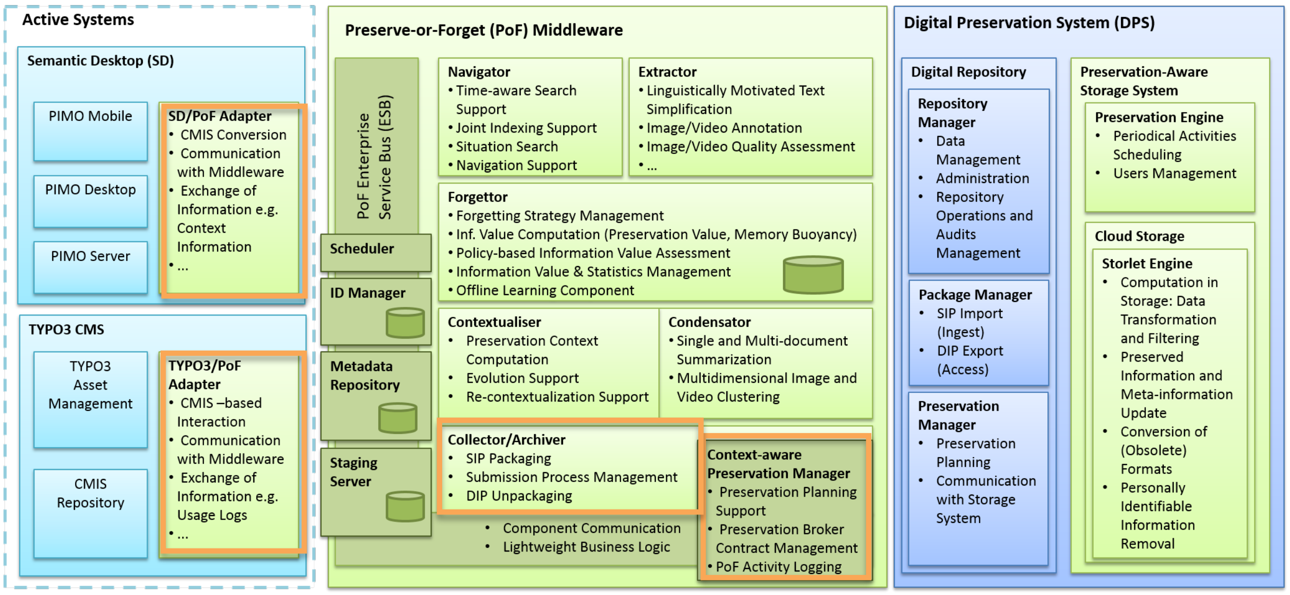
1.2 Target Audience

The Introduction section, Big Picture section, and the Summary should be of interest to most readers as it provides a brief overview of the structure of the framework built in the project and where the components described in this report fit in. The Workflow section and Implementation section are intended for those with interest in more of the details surrounding the work.

2 The Big Picture

To describe where the work reported in this deliverable fit in to the rest of the project, an overview of the role of the components and the workflows is provided here.

This workpackage is responsible for two components in the Preserve-or-Forget (PoF) middleware, as well as the PoF Adapters in the Active Systems. The overall architecture can be seen in Figure 1 with the relevant components highlighted.



Active Systems
Semantic Desktop (SD)
PIMO Mobile
PIMO Desktop
PIMO Server
SD/PoF Adapter
• CMIS Conversion
• Communication with Middleware
• Exchange of Information e.g. Context Information
• ...
TYPO3 CMS
TYPO3 Asset Management
CMIS Repository
TYPO3/PoF Adapter
• CMIS –based Interaction
• Communication with Middleware
• Exchange of Information e.g. Usage Logs
• ...
Preserve-or-Forget (PoF) Middleware
PoF Enterprise Service Bus (ESB)
Scheduler
ID Manager
Metadata Repository
Staging Server
Navigator
• Time-aware Search Support
• Joint Indexing Support
• Situation Search
• Navigation Support
Extractor
• Linguistically Motivated Text Simplification
• Image/Video Annotation
• Image/Video Quality Assessment
• ...
Forgettor
• Forgetting Strategy Management
• Inf. Value Computation (Preservation Value, Memory Buoyancy)
• Policy-based Information Value Assessment
• Information Value & Statistics Management
• Offline Learning Component
Contextualiser
• Preservation Context Computation
• Evolution Support
• Re-contextualization Support
Condensator
• Single and Multi-document Summarization
• Multidimensional Image and Video Clustering
Collector/Archiver
• SIP Packaging
• Submission Process Management
• DIP Unpackaging
• Component Communication
• Lightweight Business Logic
Context-aware Preservation Manager
• Preservation Planning Support
• Preservation Broker Contract Management
• PoF Activity Logging
Digital Preservation System (DPS)
Digital Repository
Repository Manager
• Data Management
• Administration
• Repository Operations and Audits Management
Package Manager
• SIP Import (Ingest)
• DIP Export (Access)
Preservation Manager
• Preservation Planning
• Communication with Storage System
Preservation-Aware Storage System
Preservation Engine
• Periodical Activities Scheduling
• Users Management
Cloud Storage
Storlet Engine
• Computation in Storage: Data Transformation and Filtering
• Preserved Information and Meta-information Update
• Conversion of (Obsolete) Formats
• Personally Identifiable Information Removal

Figure 1: The Collector/Archiver, the Context-aware Preservation Manager, and the Active System Adapters in the ForgetIT architecture

While Figure 2 depict an overall interaction between an Active System and a Digital Preservation System, including the need for exchange of administrative information, earlier deliverables, in particular the *Preserve-or-Forget Framework* deliverables D8.3 [Gallo et al., 2014] and D8.4 [Gallo et al., 2015b], describe the communication between the Active System adapters and the middleware and its components. The communication adopts a REST[1] approach and the exchange of Digital Objects (DO) is handled using CMIS[2]. The communication between the Digital Preservation System (DPS) and the middleware also adopts REST, but not CMIS since that currently is not a common option in DPS solutions. One design issue worth noting is that the ForgetIT project works under the assumption that a customer (Active System owner) might use several digital preservation service providers, perhaps at the same time, but most certainly over time, and that the systems involved will change.

[1] Representational State Transfer
[2] Content Management Interoperability Services – http://docs.oasis-open.org/cmis/CMIS/v1.1/os/CMIS-v1.1-os.html

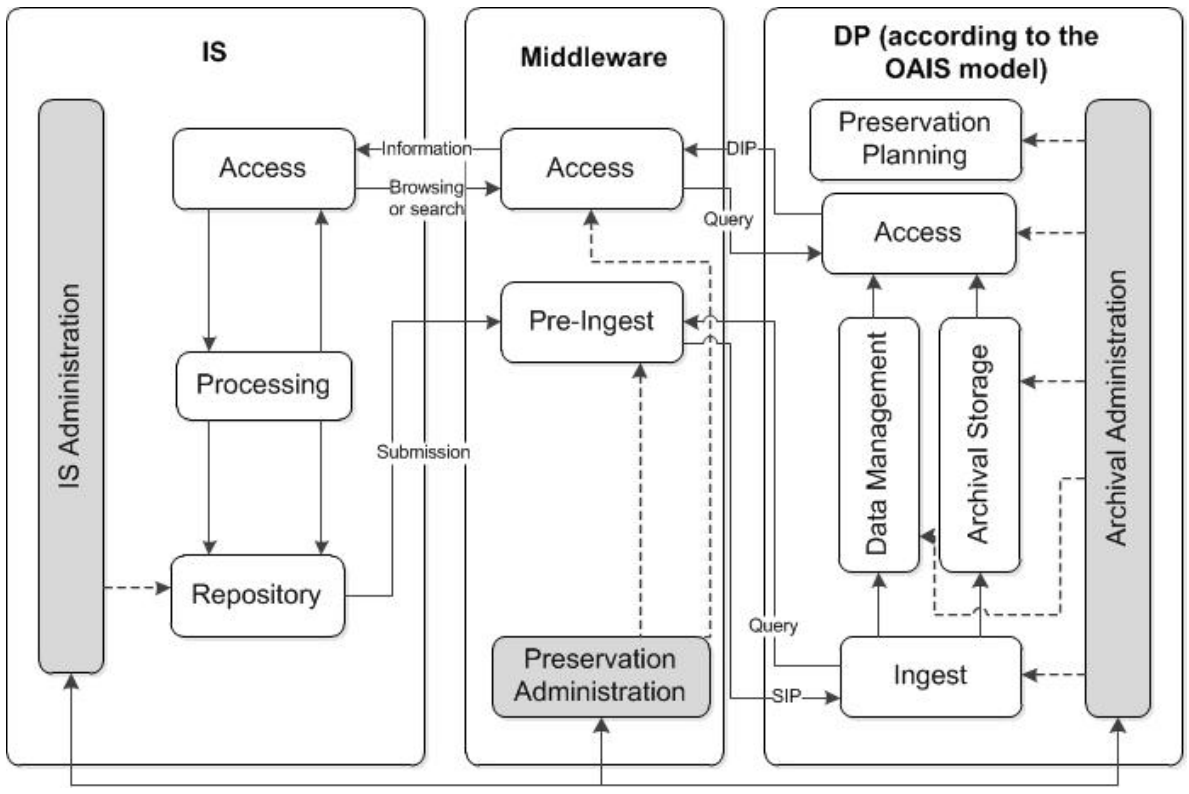


Figure 2: Model for interaction between Active System (IS) and Digital Preservation System (DP) [Afrasiabi Rad, Nilsson, Päivärinta, 2014]

2.1 Collector/Archiver software component

The Collector/Archiver can be seen as the component that handles the basic flow of digital objects between the active systems and the preservation system(s). In general it can be seen as two parts, where in a typical preservation preparation scenario, the Collector is responsible for fetching/gathering objects, that should be preserved, from the Active System and the Archiver is responsible for packaging them in a suitable way, thereafter transferring them to the DPS. In a re-activation scenario, the role of the Collector would be similar, although it would then work with the DPS and would be responsible for extraction of the Dissemination Information Package (DIP), adjusting/adapting the package according to the Preservation Broker Contract, and making the objects available to the Active System.

2.2 Context-aware Preservation Manager component

The Context-aware Preservation Manager (CaPM) is responsible for management of administrative information from the PoF middleware process. This information is used in the preservation and re-activation process to enhance the usability of digital objects when brought back from a DPS to active use in an Information System (IS). This component monitors use and change of physical and logical structures in IS, logging changes in practices; capture the use of file formats and technologies. The CaPM will use this data in the computation of change recommendations based on content value and use statistics and to propagate this information to the DPS that is responsible for keeping preserved objects usable. This component is also responsible for the establishment of a submission agreement, here called Preservation Broker Contract (PBC), to hold information that defines the terms and conditions of routes for different content acting as a semantic matchmaker between identified needs and provided digital preservation (DP) services.

2.3 Active System adapter

The active system adapter act as a bridgehead between the system that has need of preservation services and the PoF Middleware. The idea is that the middleware provide a number of standard-based interfaces for the exchange of objects and that the adapter implements that interface on the Active System side. As already mentioned, the ForgetIT project employs the CMIS standard for exchange of objects, and the Active System adapters in the two different systems implement this in two slightly different ways. The adapters are described in detail in D8.4 [Gallo et al., 2015b] but in short the TYPO3 case uses a full-fledged Alfresco[3] server acting as a CMIS repository, and the Sematic Desktop (SD) case employs an Apache Chemistry[4] implementation. Both these implementations are then (loosely) integrated with the respective systems.

2.4 Scenario

Here we introduce a scenario that is used to describe the role and functionality of certain components and functions. The purpose is to provide an informal view on the preservation process and what is involved, especially from the user perspective. In this description we assume that the "user" can be either a private person, or a person working professionally within an organisation, and we try not to make any big difference between them, although in reality user interfaces, for example, certainly could have different vocabularies and look different depending on the target group.

2.4.1 Scenario description

Establishing a contractual relationship

Initially, an agreement between the Active System user/owner and the PoF Middleware manager has to be made regarding what should be monitored for preservation. There are several things that should go into a contract and although some of them are generated as defaults, a couple of things have to be provided as input to what we refer to as a Preservation Broker Contract (PBC).

If a contact has not been established earlier, an adapter needs to be put in place in order to facilitate exchange of objects between the active system and the middleware. This adapter can be as "simple" as a CMIS server implementation that is just providing access to a certain folder on a local file system, or something more advanced with integration into the actual information system(s) producing information that should be preserved. This adapter will be referred to as an "endpoint" for the active system, and information about this needs to be put into the PBC.

The PBC also requires a name, a contact person on the active system side, and contact information. After this fairly straightforward information, the user is presented with a guided questionnaire (a "wizard") to get input on what needs the user has regarding the preservation of the objects.

Preservation Preparation

The use of contracts makes it possible to adjust the execution of preservation activities according to the evaluation of a digital collection. The preservation preparation process includes several steps that could be managed by rules stated in a contract. Rules that reflect different preservation policies dependent on content classification and purpose of use, policies that will change over time.

One important part of a the long-term management of digital documents for future understanding is to enrich the objects by adding technical, provenance, and contextual metadata. The PoF middleware demonstrate the use of services for both automated technical metadata extraction and external contextualization mechanisms that enable the linking of objects to external digitized knowledge bases and by extracting and adding information from external sources as contextual metadata D6.4 [Greenwood et al., 2016]. This is an example of a process that could be managed by a contract containing a "rule" stating when this service should execute and what kind of external source to use. Another example of a task in a preservation preparation workflow is the image duplication identification and quality analysis mechanisms [Mezaris et al., 2016] where rules stated in a contract could be reflected in the handling of objects. Other examples of the need for flexibility in the preservation process can be derived to the need for a trusted and secured management

[3] Alfresco CMIS – http://www.alfresco.com/cmis
[4] Apache Chemistry – OpenCMIS http://chemistry.apache.org/java/opencmis.html

process stating the need for execution of fixity or encryption mechanisms in the preservation preparation workflows.

These are examples of preservation activities which are part of a preservation preparation workflow that could be adjusted based on content classification or purpose of use. The possibility of adjusting preservation process activities should also be seen in light of the fact that every action and amount of data that must be handled is associated with a cost in a cloud-based service oriented preservation solution.

Preservation Action
Preservation actions are actions that are required for continuous access to a digital object. Planning of these actions is part of Preservation Planning, and often results in recommended migration pathways that incorporates identification of appropriate file formats and migration tools, but also includes aspects of security, access, storage, and that handling practices are according to policies. Decision on migration pathways should be taken with a good basis, thus having a position residing in the middle of many Active Systems and many Digital Preservation Systems gives an excellent opportunity to capture the use of file formats and type of systems in bi-directional interactions. To elaborate a bit further on this process; a technology watch mechanism, residing in every OAIS compliant DPS, obtains statistical data from the middleware that indicates that the use of a specific file format has been drastically reduced. Information that automatically could be incorporated into the administration entity in the DPS as part of an annual reporting mechanism or being triggered as an alert based on a threshold quota stated in a contract (PBC). Before taking any actions archivists, notified by the file format alert, decides to examine this further supported by the use of a graphical decision support interface that enable scrutinizing based on different statistical computations and filtering of data linked to a format registry. If a decision results in a planned migration action the contract (PBC) could define if the migration action should be executed for a specific object, and after execution how to handle the result (original and its copies) depending on different quality thresholds.

Contract changes/updates
Part of maintaining objects over time, also involves changes in both the system environment and the organisational/personal setting. This could include for example the desire to switch service provider, or circumstances that involve handing over the holdings to another individual. This means that also the contract should be able to evolve, or be replaced by a new one.

3 Context-aware Preservation Manager

The main objective of the PoF middleware is to achieve a seamless transition of digital objects in and out of ECM-based information systems (Active Systems) to OAIS-compliant preservation systems (DPS) in a many to many relationship. The Context-aware Preservation Manager (CaPM) is a component that resides inside the Preserve or Forget (PoF) middleware with the purpose of providing support to action-based components and services located in the PoF middleware by providing structure and rules to the preservation preparation and re-activation processes based on an agreement between the information Producer (Active System) and the Archive (DPS). Besides supporting activities inside the PoF middleware, the CaPM also facilitates the interactions that take place between the PoF middleware and an active system on one side and a DPS on the other. The CaPM component has been developed based on the workflow “scenarios” represented by the Preservation Preparation workflow (Figure 3), the Re-activation workflow (Figure 5), and the Setting Change Workflow (Figure 7). Based on these workflow scenarios the functional features of the CaPM has gradually evolved to a number of functional components divided into three main parts; the Preservation Broker Contract (PBC) – supporting the establishment of an contractual agreement between an active information system and a DPS. Agreements with the purpose of providing structure and rule to the relationship between information Producer and an Archive. Another component part of the CaPM is the Preservation Planning Support (PPS) that exposes the bi-directional communication between active systems and DPSs by making use of the logging mechanisms of the Collector/Archiver component. The third CaPM component is the Activity Logging (AL) mechanism that provides the ability to support a trustworthy management of activities executed in the PoF middleware by keeping track of every “action/event” that is executed. This section continues with the description of the background and motivation followed by general functional descriptions of the three sections of the CaPM; the Preservation Broker Contract (PBC), the Preserve or Forget Activity Logging (AL), and the Preservation Planning Support (PPS).

3.1 Background

The foundation for the design characteristics of the Context-aware Preservation Manager (CaPM) component is based on a thorough literature review. This identified the need for support for content-based information systems to manage an increasing amount of digital information that needs to be preserved [Päivärinta et al., 2014]. Päivärinta also identified a gap between content-based information management systems and OAIS-compliant [CCSDS, 2012] Digital Preservation Systems (DPS) that implied the need for a more aligned view on the management, producer and information consumer roles in order to interact smoothly. The paper of Päivärinta et al. [2014] identified challenges and issues in administering the integration of content-based IS and DPS to support seamless content workflow scenarios. One of these design issues was to emphasize the adaptation of preservation needs to adequate cloud based DP (Digital Preservation) services. Afrasiabi Rad, Nilsson & Päivärinta [2014] suggested a middleware [Linthicum, 2000] to be developed and placed between DP services and IS to support pre-ingest, post-access, and preservation administration.

Pre-ingest is defined as preservation preparation tasks before ingest to a DPS, post-access as tasks related to request of objects from a DPS and preparation of these objects for re-activation in an IS, and preservation administration as supportive tasks for managing the pre-ingest and post-access workflows between IS and DPS. Pre-ingest could further be elaborated as the process where digital objects are evaluated and prepared for compliance with repository standards, according to content and metadata requirements. A common pre-ingest approach is to provide a separate tool/service for the preparation of submissions; one example of such a solution has been developed as part of the HathiTrust initiative [York, 2010] that provides support for determining the standards and specifications and prepares content for ingest by modifying or transforming it to meet these specifications. A literature review by Afrasiabi Rad et al. [2014] shows that there are initiatives that have developed partial solutions that support the integration of IS and DPS, one example is part of the Safety Deposit Box (SDB), which at the time of writing this report has evolved into Preservica Enterprise Edition [Preservica, 2014], that has demonstrated its usefulness in delivery of content to the archive, but Afrasiabi Rad et al. [2014] also reveals that DP solutions that emphasize automation, close integration between IS and DPS in a many-to-many "brokered" relationship is in its infancy.

The tasks identified as needed to support the preservation administration [Afrasiabi Rad et al., 2014] should be interpreted as a translation of organizational preservation policies to DP requirements, and the mapping of these to the execution of adequate DP services. In the PoF middleware solution such DP services are there to support a seamless transition of objects from active systems (IS), automated creation of metadata, management of process execution and to handle exceptions and errors, ending with the creation and transfer of SIPs adapted to receiving DPS. Afrasiabi Rad et al., [2014] also identified the need for support for post-access activities that includes re-activation of preserved information resources back to active information systems that possibly is not the system that initially preserved them. This was the basis that influenced the start of identifying the characteristics of the Context-aware Preservation Manager (CaPM).

In the next step, and start of concretizing features of the CaPM, we have studied the result from the SHAMAN project [Wilkes et al. 2009] that identified the need for a closer integration of archiving system functionality to information system workflows (Product Life Cycle Management). Wilkes et al. [2009] identified some fundamental preservation tasks that needs to be supported based on use-cases as automated collection of metadata, tracking of relationships between objects, support of automated transformations of objects into vendor-neutral formats, the need for vendor independent access to preservation functionality, the interdependency between preservation system and specific information system, flexible support of metadata standards, monitoring of user actions during the preservation process being added as provenance history to objects, validation (completeness/correctness) of the preservation process and object before ingestion to the archive, and migration support at different stages; at pre-ingest, ingest, access, or based on a scheduled preservation plan.

One methodological framework that has an important impact on the design of CaPM is the Producer-Archive Interface Method Abstract Standard (PAIMAS) [CCSDS, 2004] which could be used as a "checklist" that provides structure to the interaction between an information producer (active system) and an archive (DPS). PAIMAS focuses on submission agreements that need to be established at pre-ingest. A submission agreement is a contract that need to be established between an active system (IS) and a DPS that specifies a data model which identifies format/data objects including any logical constructs produced by the active system and how they are represented in each transmission session [CCSDS, 2004]. Any agreement defined by PAIMAS comprises the digital objects as sizes, structures, formats, quantity, communication procedures, packaging, metadata, delivery schedules, detection of errors, and quality aspects. PAIMAS is useful as a checklist of what should be considered before any submissions to the archive have occurred but it is an "abstract standard" and is in need of concrete implementations that show how to concretize and execute an agreement.

There is also a technical recommendation that can be seen as a concretization of PAIMAS and that is the Producer-Archive Interface Specification (PAIS) [CCSDS, 2014] providing an XML-based description of data to be sent to an archive. PAIS includes a syntax for defining the digital objects along with their inter-relationship which could be aggregated into SIPs and how to instantiate this as a data package via the use of an XML Formatted Data Unit (XFDU)[5]. PAIS is useful for concretizing an agreement concerning digital object specifications and their internal relations as a basis for creating SIPs. It also provides a contribution towards agreements concerning the transfer of SIPs between active system and DPS.

However, we argue that even PAIS does not sufficiently cover the close relationship between several active systems (AS) and several DPS supported by a middleware broker solution with the aim of achieving a high level of automation. In order to achieve this we have suggested the implementation of a Preservation Broker Contract (PBC) that, in addition to providing details of the digital objects and its relationships, supports a tighter interaction pattern in a many-to-many relationship by executable service endpoint specifications, address the need for support in the appraisal of preservation candidate objects, and provide rule-based execution paths that reflects

[5] http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.116.3122&rep=rep1&type=pdf

organisational preservation policies. We also argue that an agreement between an active system and DPS should embrace the concept of service-orientation [Papazoglou, 2003] by stretching out the impact and use of DP services that goes beyond the "pre-ingest" and "ingest" procedures of a DPS; a PBC facilitate features that could translate a high-level DP strategy to be reflected in the selection and configuration of object management and storage services. In addition, we have identified that an agreement (PBC) could be useful in a re-activation process (post-access); re-contextualize copies of the preserved information resources back to active information systems by the support of restructuring of objects, definition of migration pathways, and access rights rule definitions. Besides the support of executable DP policies (rules) and to provide structural instructions by the use of a contract (PBC), the CaPM has been designed with features that support monitoring of activities and a compilation of logged objects that pass through the PoF middleware. This provides the ability to monitor middleware processes, execute alternative routes depending on the outcome and to ensure that activities are executed as agreed in the PBC. This feature also allows the statistical compilation of file formats that could be used as decision support in a preservation action planning process, deciding upon suitable migration pathways.

3.2 Support functions

The Context-aware Preservation Manager (CaPM) acts upon agreements between the information producers, consumers, and the preservation service providers by logging the digital content passing through the PoF Middleware, PoF component actions (activities), and any errors that might occur in the process of transferring content between these systems. CaPM supports the creation and uphold of submission agreements based on preservation policies, and is able to suggest actions to be taken e.g. in re-activation of content. The CaPM also have the ability to assist in preservation-planning scenarios by providing a user interface that displays graphs with statistics based on the number of objects passing through the PoF middleware per time period. The CaPM has three major subparts that support it's overall role: Preservation Broker Contract; Preserve or Forget Activity Monitoring; and Preservation Planning Support. These will be described in separate sections following below.

Preservation Broker Contract

The Preservation Broker Contract (PBC) provides structure and rules for the interactions, in either direction, between an Active System and a Digital Preservation System. In order to support the seamlessness of transitions, the PBC should contain rule-based execution paths reflecting organisational preservation policies, executable service endpoint specifications, object management configurations such as migration pathways, access rights definitions, storage options, and information needed for restructuring of objects in order to adapt to changes in the environment. Besides the machine-readable instructions, the PBC should also function as a formal agreement between the involved participants.

Preserve or Forget Activity Logging

Activity Logging supports the logging of all activities involved in the processing of objects within the PoF middleware, such as which events take place and the systems involved. This provides the ability to monitor middleware processes, execute alternative routes depending on the outcome and to document that activities are executed as agreed in the PBC. The logs also serve as input to Preservation Planning Support.

Preservation Planning Support

The Preservation Planning Support assists in the decision making processes around preservation planning. Based on input from activity logging and with information about e.g. file formats from the Collector, statistical data can be grouped by mime type classifications and filtered in different ways if desired, e.g. by systems, system types, or contracts. This information should be presented in a way that is suitable for human decision-making, e.g. some graph(s). The statistical computation of object data can also be accessed in a standardized format by a web service interface, possibly to be used for automated execution of preservation actions.

3.3 Influence on DoW Success Indicators

In the projects Description of Work, there is a success indicator for "types of major change that can be communicated to the preservation system" where the Context-aware Preservation Manager helps in keeping track of e.g. types and formats of material passing through the middleware as well as the physical and logical structure of this information. Not all of this information is relevant for the Preservation System, at least not until dissemination request is made at which time the CaPM can assist with information about the current situation and in what way the material should be brought back, including e.g. rules for transformation. The same type of information is also needed when there is a (major) change in any system involved, for example if the Active System has migrated to a new version, or perhaps even a different system.

The Preservation Broker Contract can hold rules for different activities and actions in the processing chain, which can assist the automation of these actions, thereby influencing the indicator "Seamless transition between active and archival system", as well as "Degree of automation of SIP generation" and "…automation of the transfer between archival system into active system". The establishment of a contract in itself is however hard to fully automate.

With the platform independent PBC, the PoF middleware is able to provide great flexibility in execution paths with a minimum of fixed workflow paths. Since the PBC also contains information about the systems that should be involved and loosely coupled service endpoints, the support for brokered many-to-many relationships between Active Systems and DPS has been strengthened. By formalising this in a platform independent contract, the dependency on a specific middleware solution has also decreased which, from a long-term perspective, is good since over time essentially all systems will be replaced.

In all of these processes, the logging of activities and events provide a documentation of what the objects have gone through, thereby contributing to the provenance of the objects, and a trusted management of the objects.

4 Workflow Descriptions

This section contains descriptions of the two main workflows related to the seamless transition of data between active systems and preservation systems. These workflows are for *Preservation Preparation*, *Re-activation*, and *Setting change*.

4.1 Problem Statement

In this work package, one of the objectives, as stated in the Description of Work, is to achieve "Seamless transition between active and archival system – based on forgetting methods". This includes defining workflows that describe this transition with relevant steps and involvement of components. Two major workflows are described in this section, namely the Preservation Preparation workflow, and the Re-activation workflow.

4.2 Preservation Preparation Workflow

In deliverable D8.5, *The Preserve-or-Forget Reference Model* [Gallo et al., 2016], the ideas and concepts behind the ForgetIT approach were gathered to define an implementable model. There are different layers in the model, but in Figure 3 we see the Preservation Preparation Workflow in the *Remember & Forget Layer*.

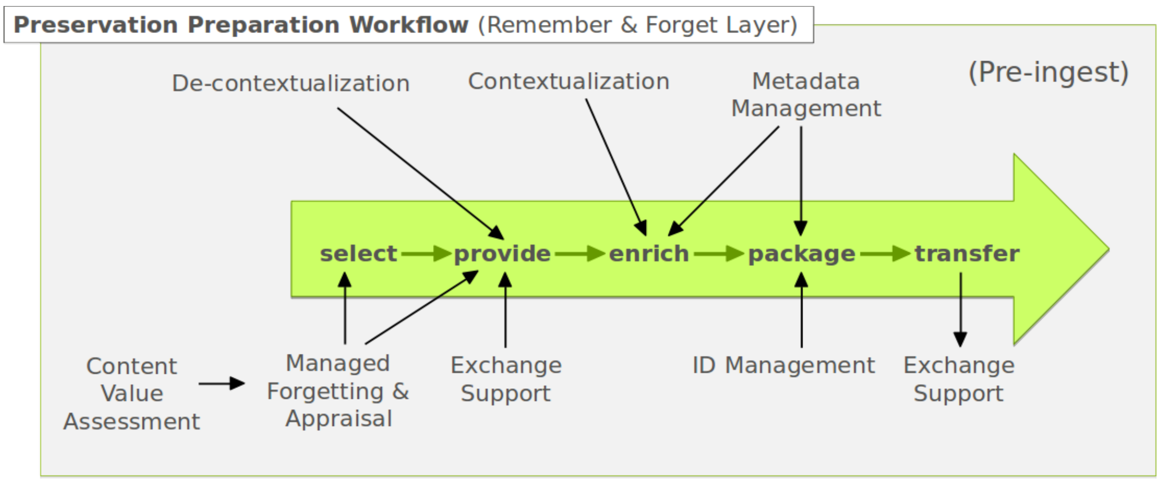


Figure 3: Preservation Preparation Workflow, reference model [Gallo et al., 2016, p. 25]

The workflow has five basic steps, *select, provide, enrich, package,* and *transfer*. Although these steps are described in more detail in D8.5 [Gallo et al., 2016], a brief description is given here. The *select* phase focuses on deciding what should be archived, and in this layer that is supported by *Managed Forgetting & Appraisal* that in turn is assisted by a *Content Value Assessment*. These two functionalities are thoroughly described in deliverable D3.3 *Strategies and Concepts for Managed Forgetting* [Kanhabua et al., 2015]. The Forgettor is the main component in this stage.

The *provide* step is supported by the *Exchange Support* which essentially is the CMIS client and the Collector, described in more detail later in this report. This step is also supported by De-contextualization, which aims at getting enough context from the active system through components developed in WP6.

In the next step, *enrich*, the selected object is enriched by the *Contextualization* functionality, which is described in detail in deliverables from WP6, the latest being D6.4 *Contextualisation Framework and Evaluation* [Greenwood et al., 2016], and by the Extractor and Condensator described in WP4 deliverables e.g D4.4 [Mezaris et al., 2016]. These components provide semantic information that should improve both the discovery and reuse of the object. This information will be treated as metadata, and therefore there is also need for a *Metadata Management* functionality, which today is mainly used for the next step in this workflow.

The *package* step is responsible for creating a suitable Submission Information Package (SIP) based on agreements between the active systems owner and the preservation system provider. The package step is supported by *Metadata Management* and *ID Management* for holding metadata that is needed for the creation of the package, and management of both local identifiers as well as identifiers received from e.g. the preservation system. The package step as well as the transfer step is largely covered by the Archiver which is described in Section 5.4.

4.2.1 Preservation Preparation workflow and the Context-aware Preservation Manager

This section contains a description of the interaction between the major components of the preservation preparation workflow. A generic workflow for the preservation preparation process has been previously described [Nilsson et al., 2015], and we therefore focus on pinpointing the role and influence of the Context-aware Preservation Manager in an implemented process workflow (Figure 4) as described in D8.4 [Gallo et al., 2015b].

In Figure 4 we have highlighted and numbered some parts that will be described below with a focus on how functionality of the Context-aware Preservation Manager can contribute to the implementation and management of such a workflow, and provide support for activity logging at each stage of the process. For details on the entire figure, we refer you to D8.4 [Gallo et al., 2015b]

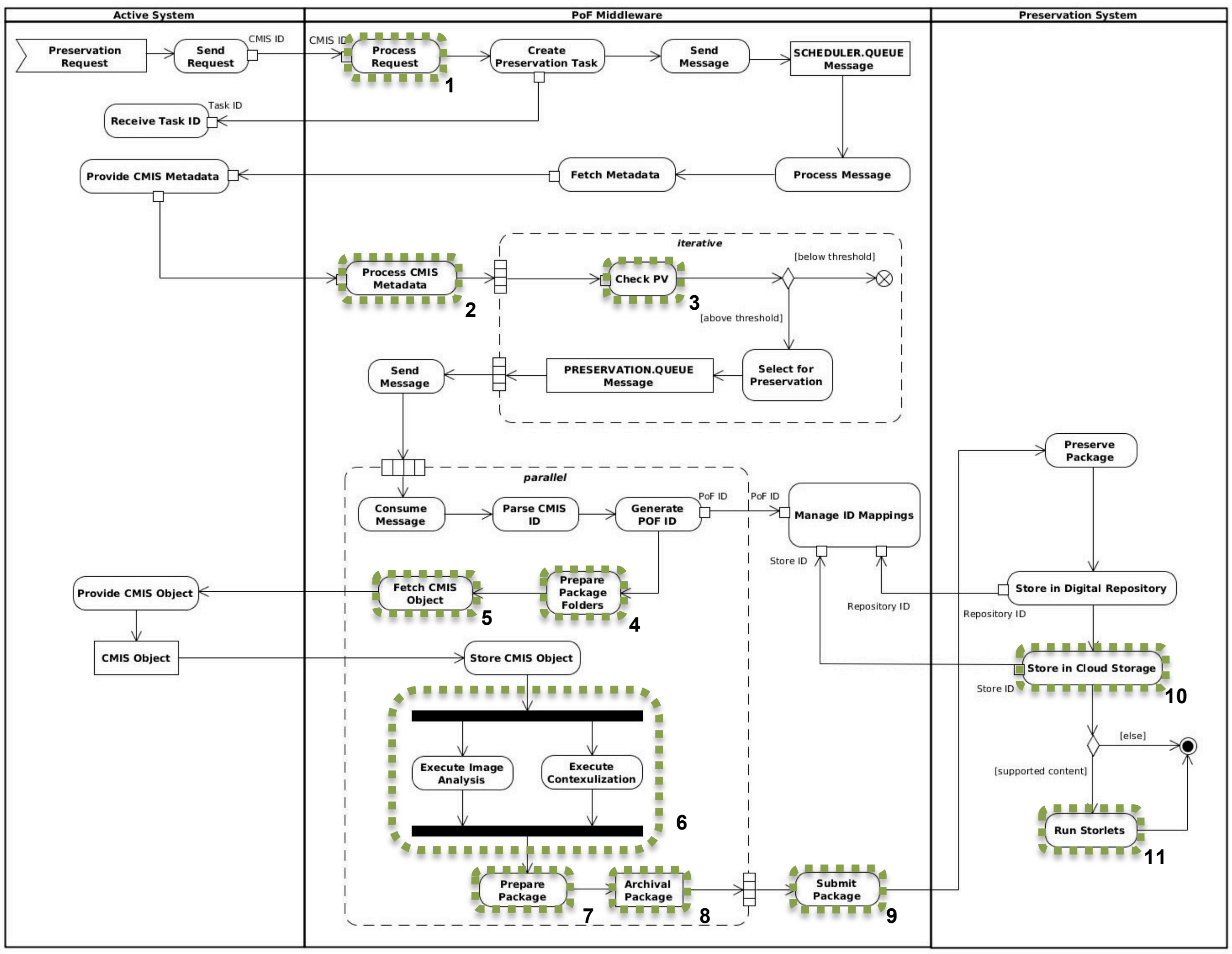


Figure 4: Activity Diagram of Preservation Preparation workflow (highlights and numbering added for this discussion) [Gallo et al., 2015b, p. 27]

1. Process Request: At this stage, if a Preservation Broker Contract ID can be provided by the Active System, the contract can already be connected to the request. This could initiate a checking of e.g. preservation storage quotas to inform the user if they are close to their limit.

2. Process CMIS Metadata: When processing the CMIS metadata fetched from the Active System, it is possible to extract information about which account the object/collection belongs to and thereby make connections to a contract, if not already provided in step 1. The contract could also include information about which metadata to process.
3. Check Preservation Value: the selection of what to preserve can be based on thresholds stated in the Preservation Broker Contract, instead of having fixed values in the Forgettor component. At this stage it is also possible to filter out objects for other reasons, also stated in the contract. As an example, the contract might state what file formats are accepted.
4. Prepare Package Folders: The Collector prepares package folders using unique identifiers. The hierarchical structuring of a folder can be described in the contract, but is currently arranged with a folder for content, and one folder for metadata (labelled "content" and "metadata" respectively).
5. Fetch CMIS Object: The Collector then fetches the objects from the Active System using connection information from the contract, and places them in the content folder. It also extracts metadata and stores it in an internal database for later processing.
6. Enrich step: When the objects have been fetched and stored in the processing area, the enrich step takes over and starts processing the objects in order to add more information and context to the objects. This includes image analysis, clustering, and contextualization handled by the Extractor and the Contextualizer respectively. The Preservation Broker Contract can here be utilised for specifying e.g. which image analysis method that should be used, and a threshold for image clustering, as well as particular external sources for context information.
7. Prepare Package: After the fetched objects have been processed by components in order to enrich the objects (e.g. the Contextualizer), the Archiver prepares the Submission Information Package by arranging the hierarchical structure of the package folders according to what is expected by the Digital Preservation System. This information should also be stated in the Preservation Broker Contract.
8. Archival Package: Based on what is agreed in the contract, the Archiver prepares the metadata for the archival package (e.g. as a combination of METS, MODS and PREMIS) and creates a submission information package.
9. Submit Package: The Preservation Broker Contract holds information on the service end-point for the Digital Preservation System (DPS), and the Archiver use this information to submit the package to the DPS.
10. Store in Cloud Storage: The Preservation Broker Contract contains information on different preservation levels and actions that are relevant for the DPS. This could include type of storage and restrictions on where (geographically) the objects are allowed to be stored.
11. Run Storlets: If needed, the Context-aware Preservation Manager can provide information from the contract to Storlets (running in the Preservation-aware Storage Service) that are about to do some processing of the objects in the DPS. This could include transformation rules and restrictions, and e.g. the handling of duplicates/redundancy.

4.3 Re-activation Workflow

As mentioned earlier, deliverable D8.5 [Gallo et al., 2016] describes the ForgetIT approach and an implementable model of this approach. In Figure 5 we see the Re-activation Workflow in the *Remember & Forget Layer*. This section focus mainly on the role of the Context-aware Preservation Manager in relation to the Re-activation Workflow, and more details on the full workflow can be found in D8.5 [Gallo et al. 2016].

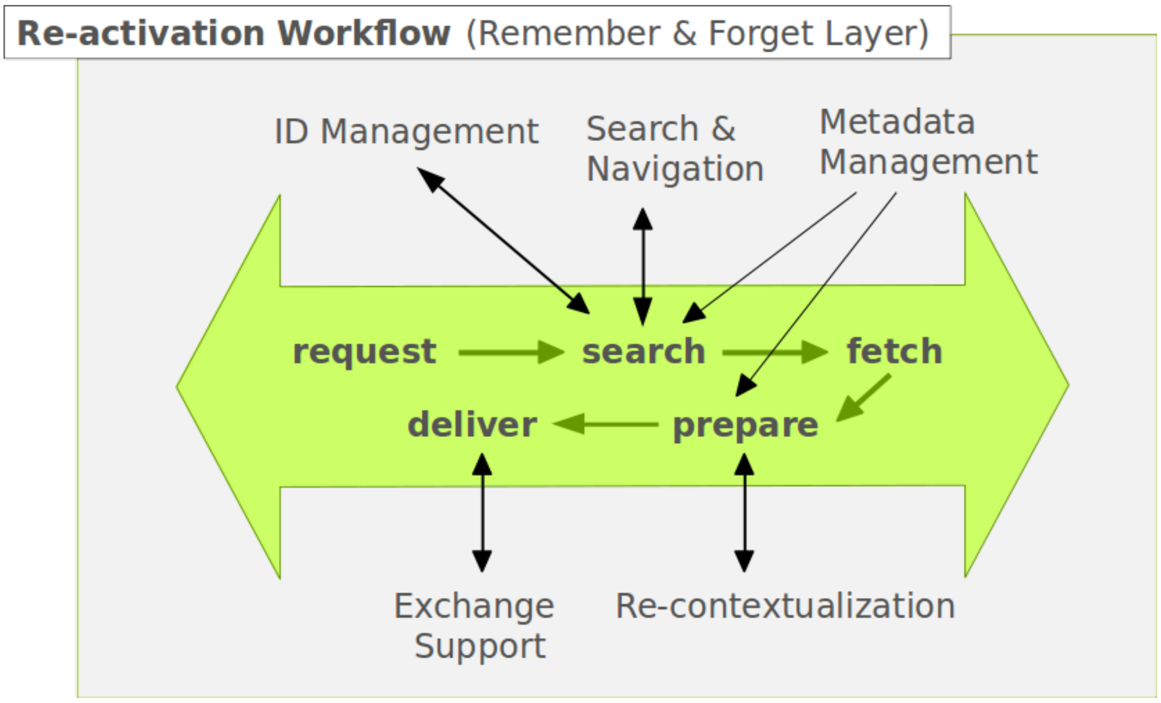


Figure 5: Re-activation Workflow, reference model [Gallo et al., 2015a, p. 24]

The workflow has five basic steps: *request; search; fetch; prepare;* and *deliver*. Although these steps are described in more detail in D8.5 [Gallo et al., 2016], a brief description is provided here.

The *request* step simply initiates the process stating that something needs to be retrieved from the preservation system. The next step, *search*, is the process of locating the object(s) in the preservation system, utilising functionality from *ID Management*, *Search & Navigation*, and *Metadata Management*. The *fetch* step is handled by exchange support, tailored to the DPS in question.

The next step is to *prepare* the fetched object(s) so that the re-activation in the Active System is as smooth as possible. This includes support from the *Re-contextualization* functionality (WP6) to potentially add semantic information, and the *Collector/Archiver* in order to prepare the package for delivery. The last step is *deliver*, which is catered for by *Exchange support*, making the package on available on a CMIS server that the Active System(s) can access.

4.3.1 Re-activation workflow and the Context-aware Preservation Manager

This workflow has been described in D5.3 (Nilsson et al., 2015) and we thereby focus on describing the parts that have been enhanced or altered by the implementation of the Context-aware Preservation Manager. Those steps are highlighted in Figure 6 and described below.

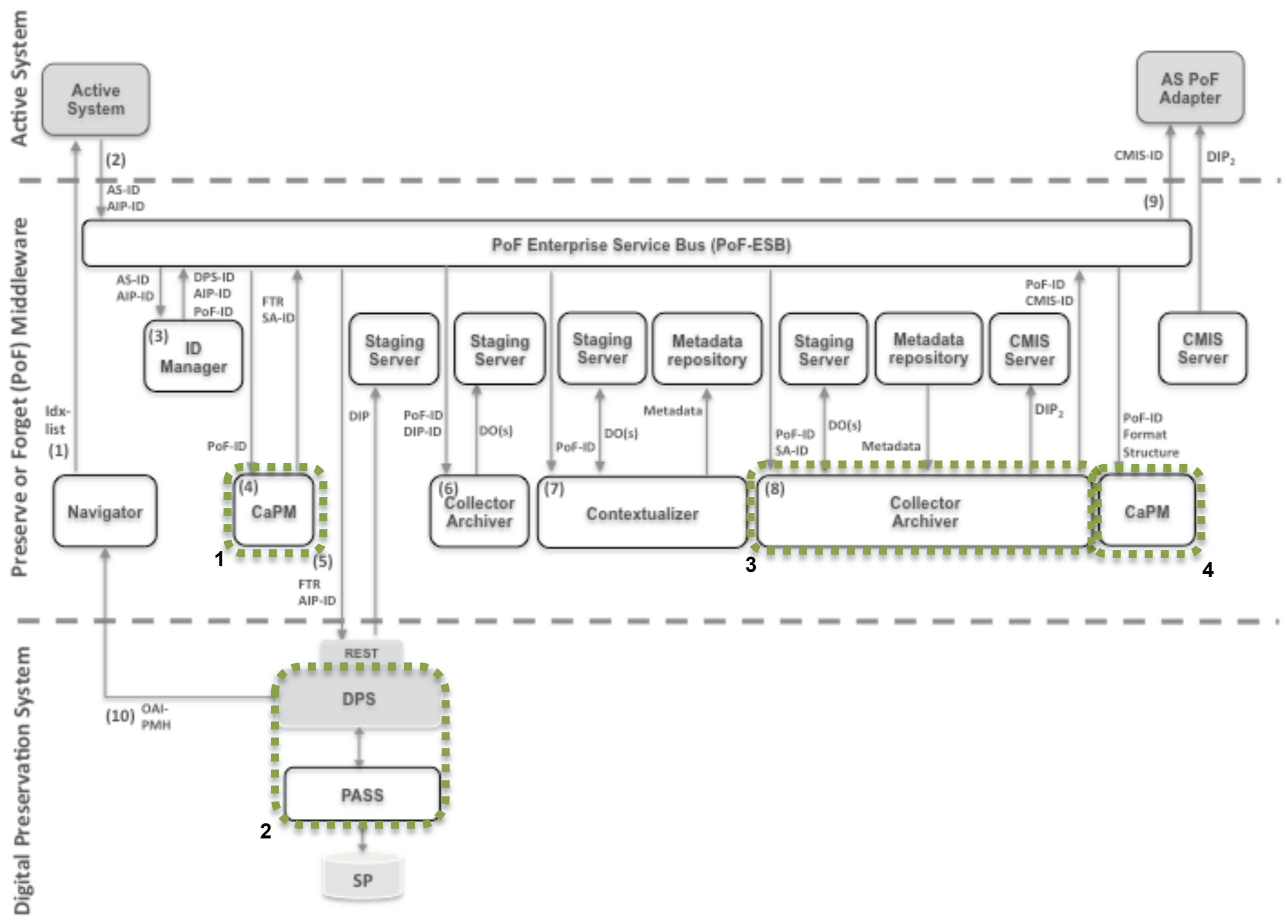


Figure 6: Re-activation Workflow and the role of the Context-aware Preservation Manager

(1) Upon re-activation, the context-aware preservation manager (CaPM) receives a request from PoF-ESB that triggers a check for the existence and return of any format transformation rules containing information about migration constraints or actions. The Preservation Broker Contract (PBC) contains information about established rules between the DPS and AS.

(2) The PoF-ESB sends a request (REST) for item(s) in the DPS identified by the AIP-ID, submitting any transformation rules that may exist. The DPS returns a dissemination information package (DIP) as response to the request. The submitted transformation rules mean that transformation might have already taken place in the DPS, before the DIP was assembled.

(3) When the package has been processed internally, e.g. by adding/updating context, the collector/archiver component starts the process of creating an adjusted dissemination information package (DIP$_2$), a package that is adapted to the receiving active system according to the Preservation Broker Contract. When the package procedure has finished, the DIP$_2$ is uploaded to an internal CMIS server. The CMIS-ID and PoF-ID is sent back to the PoF-ESB. The CaPM logs information such as use of file format and physical/logical structure from the re-activation process.

(4) The PoF-ESB notifies the AS that the package is ready, triggering the process of retrieving the DIP$_2$. The CaPM logs that this activity has taken place, and the result thereof, both for statistical use as well as traceability.

4.4 Setting Change Workflow

The Setting Change workflow consists of four different phases with two different starting points (as depicted in Figure 7): the *activity monitoring* which logs the bi-directional communication between the Active System and DPS including process activities, systems in action, and digital objects passing through, the *change assessment* that detects and propagate change in usage, the *change estimation* suggests suitable change recommendations based on rules defined in Preservation contract including e.g. preservation value, and use statistics. The *change recommendation* propagates recommended actions to DPS, which could be of different types, such as transformation of content or change of physical and logical content structure.

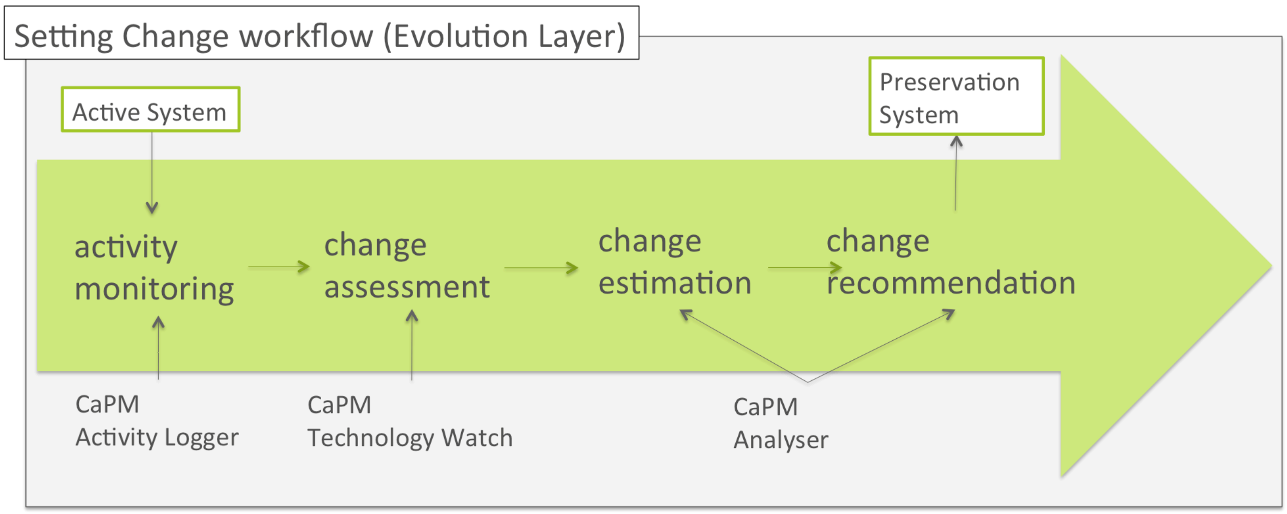


Figure 7: Setting Change Workflow, reference model [Gallo et al. 2016, p. 31]

CaPM Activity Logger: The Activity Logger provides support for monitoring of activities in PoF workflows. A typical example is to keep track of the execution of tasks executed by components in the workflows. This data will then serve as one input to the change assessment.

CaPM Technology watch: The Technology Watch gathers data on objects handled by the PoF workflows. This includes information on Active System and DPS, as well as technical metadata on the object e.g. file format. This information, combined with input from Activity Logger, forms the basis for a change assessment which if needed triggers a change estimation. There are several things to consider in "technology watch". The component could trigger an alert based on thresholds related to storage quota, usage of file formats grouped by year, or changes of the actual systems (Active System, PoF, Digital Preservation System).

CaPM Analyser: Based on an initial assessment of the need for a setting change, the change estimation process is supported by the Analyser, which aggregates and processes the data provided from earlier functions and provides a visual interface for human interpretation. The Analyser may calculate a preferred setting change based on thresholds and rules, provided by e.g. Preservation Contract, but these can be overridden via a user through the graphical interface. The decision, be it automatic or manual, will then become a change recommendation that is handed over to the DPS for implementation.

4.5 Influence on DoW Success/Progress Indicators

As described in the DoW of the project, the performance indicator related to workflow states: "Seamless transition between active and archival system – based on forgetting methods". This seamless transition is not entirely related to components in WP5, especially not with regard to the second part, forgetting methods. The workflows described do however involve many components outside of WP5, as intended, and by employing an Enterprise Service Bus together with the active system adapters, the exchange support, as well as preparation of information packages for both preservation preparation and re-activation (ingest and access), the transition is, albeit not fully seamless, nearly seamless. This to some extent also is dependent on how close the integration between Active System Adapters and the Active Systems are.

The functionality mentioned above is supported by the Preservation Broker Contract that can hold rules for different activities and actions in the processing chain, which assist the automation of these actions.

5 Implementation Details

This section describe in some detail the component implementations, dependencies and "behind the scene" interactions. It starts with a brief problem statement and then describes components developed in WP5 and their relation to the workflows. The simplified component diagrams in this section depict the WP5 components and the most relevant relations to other components or modules; use of external tools and components (i.e. components not built in the project) are not included in the component diagrams. The use of classes in the component diagrams are described in each functional section. This section is mainly intended for readers interested in implementation details and considerations.

5.1 Problem Statement

The overarching objective for WP5 is to facilitate "smooth bi-directional transition between information management and preservation". This requires a fair amount of automation and normalization of communication and exchange of digital objects. The components implemented in WP5 mainly focus on exchange of digital objects, and communication of changes in the environment, mainly to the digital preservation system.

5.2 Common Details

Unless otherwise stated, the components described in this chapter have many things in common which are described here. Any additions or diversions are described under respective section for the individual components.

API and I/O Formats The components are written in Java and makes use of available technologies. For the implementation of a permanent store of data the components uses MySQL**Error! Bookmark not defined.** as an RDBMS[6](DB) and JDBC[7] for communication. The interaction with data stored in XML is done by the use of a simple JSON API for XML**Error! Bookmark not defined.** and Jersey, a RESTful API (JAX-RS)**Error! Bookmark not defined.** is used for the http-based interfaces. The logging mechanism is supported by the Apache log4j API**Error! Bookmark not defined.**. The communication between active system(s) and the `CMISapp` is implemented using the OpenCMIS Client API[8].

5.3 Collector

Component Role The *Collector* is the name of a group of components depicted in Figure 8 that are invoked by the PoF middleware (Enterprise Service Bus) when digital objects and metadata need to be collected from the active system(s). The `Collector` is triggered by a call to its REST API**Error! Bookmark not defined.** submitting a PBC-ID that identifies the Preservation Broker Contract (PBC) and the unique PoF-ID that bundles activities that belongs to same preservation preparation process. The `Collector` starts the process by the use of the `CMISapp` class as an adapter to fetch objects (files) and metadata from the active system and puts them in a dedicated space, labelled with the unique PoF-ID, on the staging server. Before the `Collector` retrieved any object it could verify that fetched objects are in line with the agreement between the information Producer and the Archive, stated in the PBC. When objects (files) have been fetched the `FillMets` class is triggered and starts to fetch metadata about the objects from the active system, a process also supported by the `CMISapp.` The `FillMets` class saves fetched metadata in a database stored in the PoF Middleware and then initiates the packaging process (see 5.4) that will use the output from the `Collector` when creating a Submission Information Package (SIP). The Collector uses the Preservation Broker Contract (PBC) to get hold of structure information and rules during the preservation preparation process as described in section (5.6.1).

WP and Deliverables The Collector component has been developed within WP5 (Joint Information and Preservation Management). This is the second release of the component that was

[6] https://en.wikipedia.org/wiki/Relational_database_management_system
[7] http://www.oracle.com/technetwork/java/overview-141217.html
[8] http://chemistry.apache.org/java/developing/guide.html

previous described in deliverable D5.3 [Nilsson et al., 2015] the contributing partners are mainly LTU and EURIX.

API and I/O Formats The Collector component utilises common APIs as described under section 5.2. The class diagram in Figure 8 does not include "helper" classes, it includes only the major classes of the Collector and is divided into three separate software packages (jar-files). The `Collector` is part of the `CollectorArchiverIngest.jar`, the `CMISapp` is part of `mavenCMISclent.jar` and the `FillMets` class is part of the `ModsTblFill.jar`.

Status and Workplan The current version of the Collector provides all expected functionalities, although further improvements are planned for work outside of this project. This would include extended interaction with the CaPM and (PoF) middleware.

License The source code of this component is released as Open Source, as part of the PoF Middleware code.

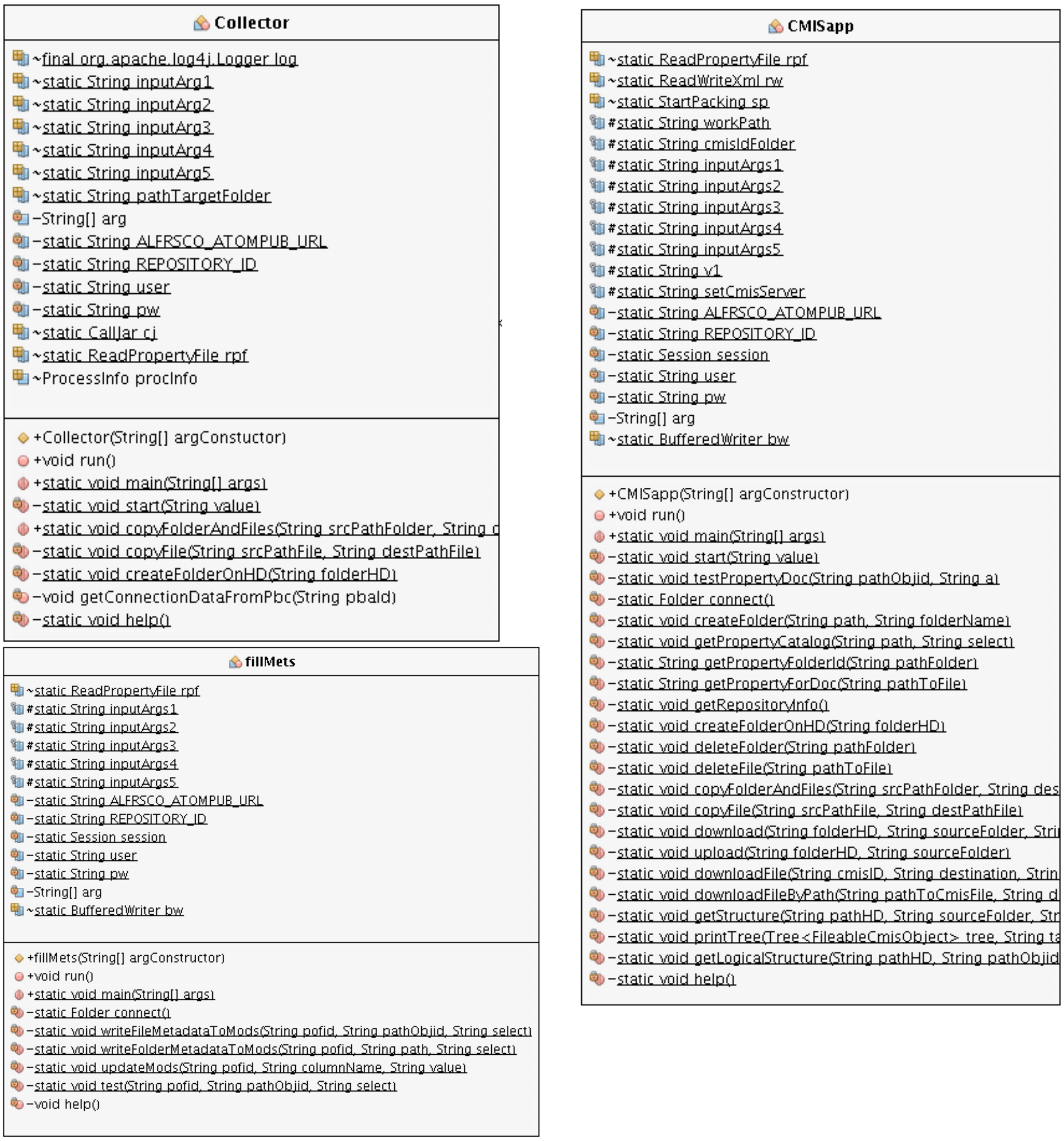

Collector
~final org.apache.log4j.Logger log
~static String inputArg1
~static String inputArg2
~static String inputArg3
~static String inputArg4
~static String inputArg5
~static String pathTargetFolder
−String[] arg
−static String ALFRSCO_ATOMPUB_URL
−static String REPOSITORY_ID
−static String user
−static String pw
~static CallJar cj
~static ReadPropertyFile rpf
~ProcessInfo procInfo
+Collector(String[] argConstuctor)
+void run()
+static void main(String[] args)
−static void start(String value)
+static void copyFolderAndFiles(String srcPathFolder, String d
−static void copyFile(String srcPathFile, String destPathFile)
−static void createFolderOnHD(String folderHD)
−void getConnectionDataFromPbc(String pbaId)
−static void help()
fillMets
~static ReadPropertyFile rpf
#static String inputArgs1
#static String inputArgs2
#static String inputArgs3
#static String inputArgs4
#static String inputArgs5
−static String ALFRSCO_ATOMPUB_URL
−static String REPOSITORY_ID
−static Session session
−static String user
−static String pw
−String[] arg
~static BufferedWriter bw
+fillMets(String[] argConstructor)
+void run()
+static void main(String[] args)
−static Folder connect()
−static void writeFileMetadataToMods(String pofid, String pathObjid, String select)
−static void writeFolderMetadataToMods(String pofid, String path, String select)
−static void updateMods(String pofid, String columnName, String value)
−static void test(String pofid, String pathObjid, String select)
−void help()
CMISapp
~static ReadPropertyFile rpf
~static ReadWriteXml rw
~static StartPacking sp
#static String workPath
#static String cmisIdFolder
#static String inputArgs1
#static String inputArgs2
#static String inputArgs3
#static String inputArgs4
#static String inputArgs5
#static String v1
#static String setCmisServer
−static String ALFRSCO_ATOMPUB_URL
−static String REPOSITORY_ID
−static Session session
−static String user
−static String pw
−String[] arg
~static BufferedWriter bw
+CMISapp(String[] argConstructor)
+void run()
+static void main(String[] args)
−static void start(String value)
−static void testPropertyDoc(String pathObjid, String a)
−static Folder connect()
−static void createFolder(String path, String folderName)
−static void getPropertyCatalog(String path, String select)
−static String getPropertyFolderId(String pathFolder)
−static String getPropertyForDoc(String pathToFile)
−static void getRepositoryInfo()
−static void createFolderOnHD(String folderHD)
−static void deleteFolder(String pathFolder)
−static void deleteFile(String pathToFile)
−static void copyFolderAndFiles(String srcPathFolder, String des
−static void copyFile(String srcPathFile, String destPathFile)
−static void download(String folderHD, String sourceFolder, Stri
−static void upload(String folderHD, String sourceFolder)
−static void downloadFile(String cmisID, String destination, Strin
−static void downloadFileByPath(String pathToCmisFile, String d
−static void getStructure(String pathHD, String sourceFolder, Str
−static void printTree(Tree<FileableCmisObject> tree, String ta
−static void getLogicalStructure(String pathHD, String pathObjid
−static void help()

Figure 8: Class diagram for Collector component

5.4 Archiver

Component Role The *Archiver* is the name of a group of components, depicted in Figure 9 that are designed for the creation of Submission Information Packages (SIPs)[9]. The PoF Middleware manager triggers the `ForgetITsip` class to; create the package structure (content, metadata, and system folders), generate a fixity value (hash sum) for each object (file) supported by the `HashSum` class, and continue with the identification of file format (MIME), extraction of technical metadata from each object, saving it in a database managed by the `Droid` class. The `ForgetITsip` adds generated contextualization metadata files [Greenwood et al., 2016] to the metadata folder and the `MetsCreator` class generates the metadata files (METS, MODS etc.), that describes the content of the SIP, adapted for the ingest mechanism at the receiving digital preservation system (DPS). Before the Archiver initiates a transfer to a receiving preservation system it uses the `CreateTarFile` class to compress the SIP to a tar or zip package. During the process of creating a SIP the Archiver uses the PoF-ID to keep track of the activities and objects that are related to a specific preservation preparation process. The Archiver uses the Preservation Broker Contract (PBC) to get hold of structure information and rules during the preservation preparation process as described in section 5.7.1.

API and I/O Formats The Archiver component utilises common APIs as described under section 5.2 with the following additions. The compression of the SIP is supported by Apache Commons Compress API[10]. The `MetsCreator` uses the METS API[11] and file identification is based on the use of DROID[12]. The creation of fixity checksums is executed by the support of md5deep[13]. Figure 9 depicts the class diagram without “helper” classes, it includes only the major classes of the Archiver and build to the `ForgetITsip.jar` software package.

Status and Workplan The current version of the Archiver provides all expected functionalities although further improvements are planned for work outside of this project. This would include extended interaction with the CaPM and (PoF) middleware.

License The source code of this component is released as Open Source, as part of the PoF Middleware code.

[9] http://documents.clockss.org/index.php/Definition_of_SIP
[10] https://commons.apache.org/proper/commons-compress/
[11] http://hul.harvard.edu/mets/
[12] http://www.nationalarchives.gov.uk/information-management/manage-information/preserving-digital-records/droid/
[13] https://en.wikipedia.org/wiki/Md5deep

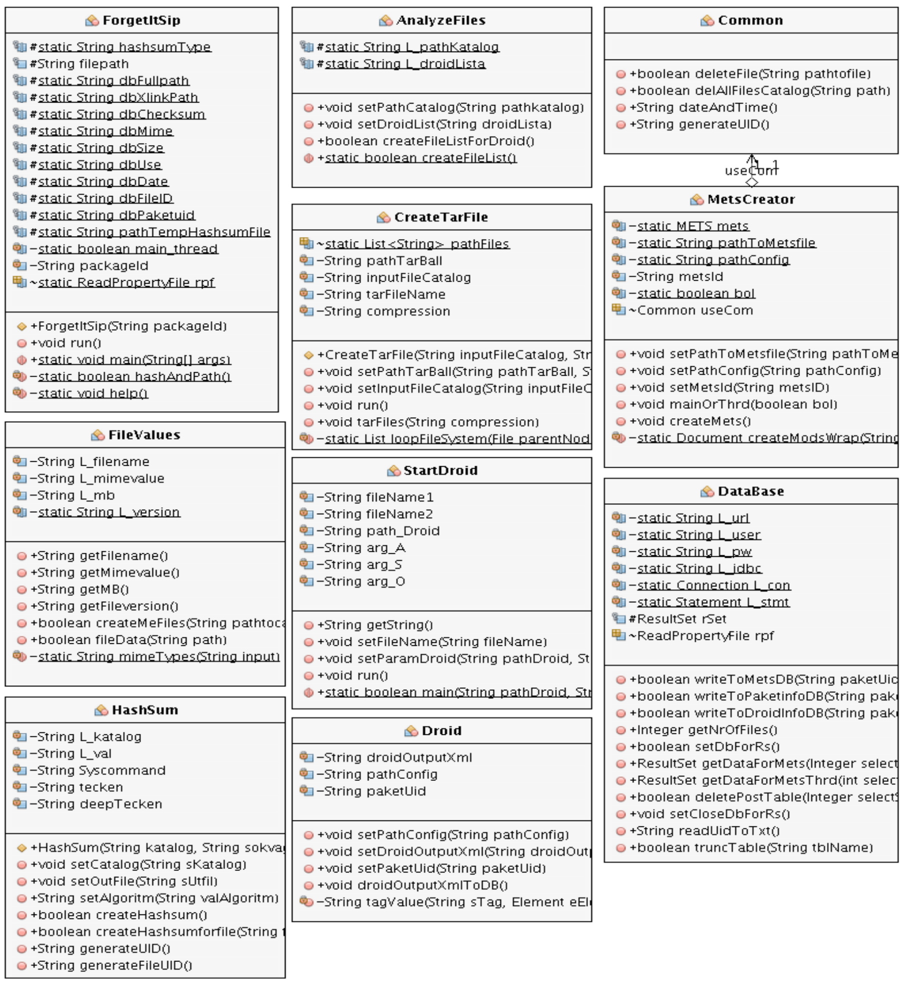

ForgetItSip
#static String hashsumType
#String filepath
#static String dbFullpath
#static String dbXlinkPath
#static String dbChecksum
#static String dbMime
#static String dbSize
#static String dbUse
#static String dbDate
#static String dbFileID
#static String dbPaketuid
#static String pathTempHashsumFile
-static boolean main_thread
-String packageId
~static ReadPropertyFile rpf
+ForgetItSip(String packageId)
+void run()
+static void main(String[] args)
-static boolean hashAndPath()
-static void help()
AnalyzeFiles
#static String L_pathKatalog
#static String L_droidLista
+void setPathCatalog(String pathkatalog)
+void setDroidList(String droidLista)
+boolean createFileListForDroid()
+static boolean createFileList()
Common
+boolean deleteFile(String pathtofile)
+boolean delAllFilesCatalog(String path)
+String dateAndTime()
+String generateUID()
useCom
1
MetsCreator
-static METS mets
-static String pathToMetsfile
-static String pathConfig
-String metsId
-static boolean bol
~Common useCom
+void setPathToMetsfile(String pathToMe
+void setPathConfig(String pathConfig)
+void setMetsId(String metsID)
+void mainOrThrd(boolean bol)
+void createMets()
-static Document createModsWrap(String
CreateTarFile
~static List<String> pathFiles
-String pathTarBall
-String inputFileCatalog
-String tarFileName
-String compression
+CreateTarFile(String inputFileCatalog, Str
+void setPathTarBall(String pathTarBall, S
+void setInputFileCatalog(String inputFileC
+void run()
+void tarFiles(String compression)
-static List loopFileSystem(File parentNod
FileValues
-String L_filename
-String L_mimevalue
-String L_mb
-static String L_version
+String getFilename()
+String getMimevalue()
+String getMB()
+String getFileversion()
+boolean createMeFiles(String pathtoc
+boolean fileData(String path)
-static String mimeTypes(String input)
StartDroid
-String fileName1
-String fileName2
-String path_Droid
-String arg_A
-String arg_S
-String arg_O
+String getString()
+void setFileName(String fileName)
+void setParamDroid(String pathDroid, St
+void run()
+static boolean main(String pathDroid, Str
DataBase
-static String L_url
-static String L_user
-static String L_pw
-static String L_jdbc
-static Connection L_con
-static Statement L_stmt
#ResultSet rSet
~ReadPropertyFile rpf
+boolean writeToMetsDB(String paketUic
+boolean writeToPaketinfoDB(String pak
+boolean writeToDroidInfoDB(String pak
+Integer getNrOfFiles()
+boolean setDbForRs()
+ResultSet getDataForMets(Integer select
+ResultSet getDataForMetsThrd(int selec
+boolean deletePostTable(Integer select
+void setCloseDbForRs()
+String readUidToTxt()
+boolean truncTable(String tblName)
HashSum
-String L_katalog
-String L_val
-String Syscommand
-String tecken
-String deepTecken
+HashSum(String katalog, String sokva
+void setCatalog(String sKatalog)
+void setOutFile(String sUtfil)
+String setAlgoritm(String valAlgoritm)
+boolean createHashsum()
+boolean createHashsumforfile(String
+String generateUID()
+String generateFileUID()
Droid
-String droidOutputXml
-String pathConfig
-String paketUid
+void setPathConfig(String pathConfig)
+void setDroidOutputXml(String droidOut
+void setPaketUid(String paketUid)
+void droidOutputXmlToDB()
-String tagValue(String sTag, Element eEl

Figure 9: Class diagram for Archiver component

5.5 Access

Component Role The *Access* component is a group of components based on the `Collector` and helper classes, depicted in Figure 10, that is responsible for fetching and unpacking content packages from the DPS to active systems. The re-activation process starts with unpacking a Dissemination Information Package (DIP) as retrieved from the DPS. This takes place on the *Staging Server* where the `StartUpload` class creates a folder structure on the server named by a PoF-ID. After this, `MovingFiles` moves the files into this structure. The `ZipUnzip` class executes the uncompressing procedures. There is also an option to unpack the objects into a folder structure defined by an XML file (created at pre-ingest to reflect the active system structure). After this, other middleware components can process the objects if needed. The content and metadata is then exposed in the resulting folder by a CMIS server that enables access for the active system.

API and I/O Formats The Access component utilises common APIs as described under section 5.2 with the addition of Apache Commons Compress API supporting the uncompressing of DIP. The classes making up the Access component are built to the `CollectorArchiverAccess.jar` software package.

Status and Workplan The use of the Access component in the re-activation workflow is limited but it provides the expected functionality of unpacking and restructure of content to folder structures according to instructions. Further improvements are planned for work outside of this project, which would include extended interaction with the CaPM and (PoF) middleware.

License The source code of this component is released as Open Source, as part of the PoF Middleware code.

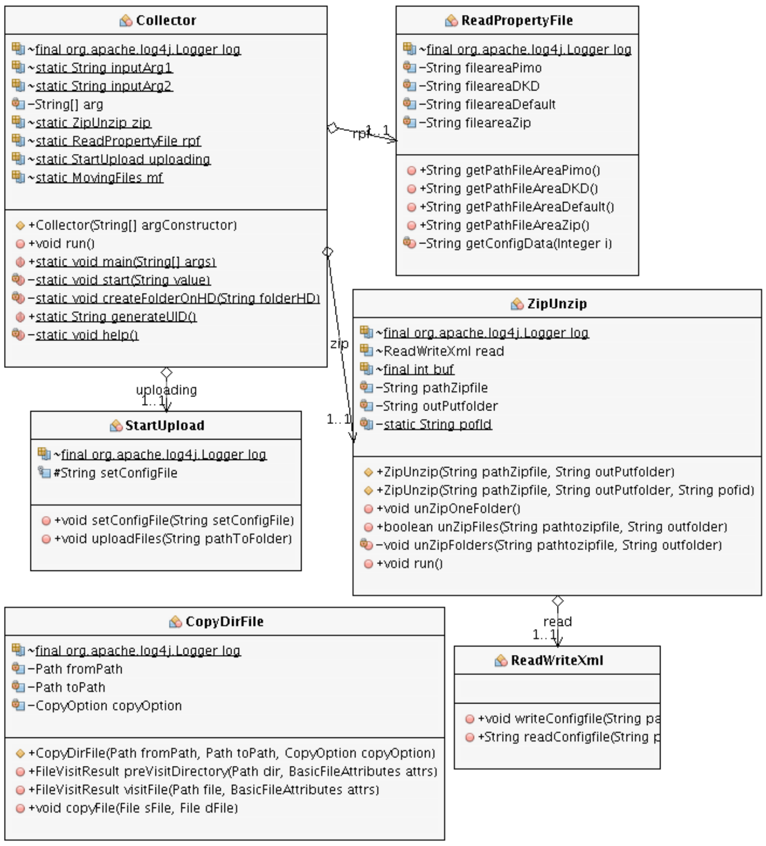

Collector
~final org.apache.log4j.Logger log
~static String inputArg1
~static String inputArg2
−String[] arg
~static ZipUnzip zip
~static ReadPropertyFile rpf
~static StartUpload uploading
~static MovingFiles mf
+Collector(String[] argConstructor)
+void run()
+static void main(String[] args)
−static void start(String value)
−static void createFolderOnHD(String folderHD)
+static String generateUID()
−static void help()
ReadPropertyFile
~final org.apache.log4j.Logger log
−String fileareaPimo
−String fileareaDKD
−String fileareaDefault
−String fileareaZip
+String getPathFileAreaPimo()
+String getPathFileAreaDKD()
+String getPathFileAreaDefault()
+String getPathFileAreaZip()
−String getConfigData(Integer i)
rpf
1..1
zip
1..1
uploading
1..1
StartUpload
~final org.apache.log4j.Logger log
#String setConfigFile
+void setConfigFile(String setConfigFile)
+void uploadFiles(String pathToFolder)
ZipUnzip
~final org.apache.log4j.Logger log
~ReadWriteXml read
~final int buf
−String pathZipfile
−String outPutfolder
−static String pofId
+ZipUnzip(String pathZipfile, String outPutfolder)
+ZipUnzip(String pathZipfile, String outPutfolder, String pofid)
+void unZipOneFolder()
+boolean unZipFiles(String pathtozipfile, String outfolder)
−void unZipFolders(String pathtozipfile, String outfolder)
+void run()
read
1..1
CopyDirFile
~final org.apache.log4j.Logger log
−Path fromPath
−Path toPath
−CopyOption copyOption
+CopyDirFile(Path fromPath, Path toPath, CopyOption copyOption)
+FileVisitResult preVisitDirectory(Path dir, BasicFileAttributes attrs)
+FileVisitResult visitFile(Path file, BasicFileAttributes attrs)
+void copyFile(File sFile, File dFile)
ReadWriteXml
+void writeConfigfile(String pa
+String readConfigfile(String p

Figure 10: Class diagram for the Access component

5.6 *Collector/Archiver/Access-RESTful services*

REST APIs are published using Jersey**Error! Bookmark not defined.**, the reference implementation of JAX-RS specification for RESTful web services. In the following we list the available APIs with the expected parameters and the output format. Table 1 shows the common REST APIs for the CaPM-PBC component.

Server path	/forgetit/restservice/collectorarchiver/cmis
Supported response types	TEXT
HTTP request type	GET
Description	RESTful API for the Collector/Archiver component

Table 1: Collector/Archiver-REST common interface

5.6.1 Collector-RESTful-API

Table 2 shows a selected subset of RESTful services exposing information from the Collector component. The selection has been made based on the importance of providing basic functionality.

Request path	/downloadfile/{pimo11}/{abx-123-Aa}/{pimo:1427296226816:8}/{objid}/{pbc-1}
Response	`downloadfile, pimo11, abx-123-Aa, pimo:1427296226816:8, pbc-1`
Description	**Parameters:**downloadfile= path to internal method, pimo11=internal identification of CMIS-server, abx-123-Aa=unique internal process identifier (PoF-ID), pimo:1427296226816:8=identification of a single object exposed by the adapter (CMIS-object-Id),objid=instruction stating that the object to be fetched is identified by an object-id, pbc-1=preservation broker contract identifier. **Returns information**: submitted parameters as confirmation for request **Example of use:** used by the PoF middleware manager for triggering the Collector component to fetch an object from the active system, identified by its id, exposed by a CMIS adapter. This process includes the construction of a folder structure with a parent folder named by the PoF-ID and the underlying folders content, metadata, and system. It saves the downloaded object in the content folder and other descriptive info in the metadata folder.
Request path	/download/{EURIX}/{abx-124-Ab}/{workspace://SpacesStore/c85123e5-39c3-4fb2-b8ae-ed12d538ca67}/{objid}/{pbc-1}
Response	`download, EURIX, abx-124-Ab`
Description	**Parameters:** download= path to internal method, EURIX =internal identification of CMIS-server, abx-124-Ab=unique internal process identifier (PoF-ID), workspace://SpacesStore/c85123e5-39c3-4fb2-b8ae- ed12d538ca67=identification of a folder exposed by the adapter (CMIS-folder-Id), objid=instruction stating that the folder is identified by a folder-id, pbc-1=preservation broker contract identifier. **Returns information**: submitted parameters as confirmation for request **Example of use:** used by the PoF middleware manager for triggering the Collector component to fetch objects from a specified folder in the active system, identified by the id, exposed by a CMIS adapter. This process includes the construction of a folder structure with a parent folder named by the PoF-ID and the underlying folders content, metadata, and system. It saves the downloaded object in the content folder and other descriptive info in the metadata folder.
Request path	/{download}/{EURIX}/{abx-125-Ac}/{/content/docs/}/{path}/{pbc-1}
Response	`download, EURIX, abx-125-Ac`
Description	**Parameters:** download=path to internal method EURIX =internal identification of CMIS-server, abx-125-Ac=unique internal process identifier (PoF-ID), /content/docs/=identification of a folder exposed by the adapter (CMIS-folder-path), path=instruction stating that the folder is identified by a folder-path, pbc-1=preservation broker contract identifier. **Returns information**: submitted parameters as confirmation for request **Example of use:** used by the PoF middleware manager for triggering the Collector component to fetch objects from a specified folder in the active system, identified by the id, exposed by a CMIS adapter. This process includes the construction of a folder structure with a parent folder named by the PoF-ID and the underlying folders content, metadata, and system. It saves the downloaded object in the content folder and other descriptive (e.g. logical structure) info in the metadata folder.

Table 2: Collector-REST-API

5.6.2 Archiver-RESTful-API

Table 3 shows a subset of RESTful services exposing information from the Archiver component, based on importance for fulfilling the basic functionality needed for the preservation preparation workflow.

Request path	/updatemods/{abx- 123-Aa}/{personal}/{Francesco}/
Response	`abx-123-Aa, personal, Francesco`
Description	**Parameters:** updatemods=path to internal method, abx-123-Aa=unique internal process identifier (PoF-ID), personal=identifies column name in mods table, Francesco=update value. **Returns information**: submitted parameters as confirmation for request **Example of use:** provides the ability to update metadata as part of the SIP. Identifies the PoF process it belongs to that later becomes a SIP-Id, the column in db that holds the data, and the value.
Request path	/packageandzip/{abx-123-Aa}/
Response	`abx-123-Aa`
Description	**Parameters:** packageandzip=path to internal method, abx-123-Aa=unique internal process identifier (PoF-ID) **Returns information**: submitted parameter as confirmation for request **Example of use:** triggers the start of the process for creating a SIP as stated in a PBC e.g. creating a folder structure, a mets XML-file containing mods structure, compressing content files and metadata files to a SIP package file (zip/tar). This process is triggered by the PoF manager.

Table 3: Archiver-REST-API

5.6.3 Access-RESTful-API

Table 4 shows a subset of RESTful services exposing information from the Access component that is part of the Collector/Archiver. The selected services provide functionality important for the Re-activation workflow.

Request path	/unzippackage/{abc-321-Bb}/{dip-123-Aa.zip/
Response	`abc-321-Bb, dip-123-Aa.zip`
Description	**Parameters:** unzippackage=path to internal method, abc-321-Bb=unique internal process identifier (PoF-ID), dip-123-Aa.zip=identifies the DIP. **Returns information**: submitted parameters as confirmation for request **Example of use:** uncompress a DIP package in a folder structure identified by the PoF-ID. This process is triggered by the PoF manager. This process is triggered by the PoF manager.
Request path	/buildstructure/{abx-123-Aa}
Response	`abx-123-Aa`
Description	**Parameters:** buildstructure=path to internal method, abx-123-Aa=unique internal process identifier (PoF-ID). **Returns information**: submitted parameter as confirmation for request **Example of use:** This service is used in a scenario where there exists a physical structure file (structure.xml) in the DIP that is fetched from the preservation system. It reads the structure file, re-creates the folder structure, and moves the files to the folders according to the specification in structure.xml. If no structure.xml exists or is “out-dated” a default structure could be used. The instructions for using this feature is stated in the PBC and is triggered by the PoF manager.

Table 4: Access-REST-API

5.7 Context-aware Preservation Manager

Component Role The Context-aware Preservation Manager (Figure 11) is responsible for supporting the PoF middleware activities by the creation of a submission agreement, a Preservation Broker Contract (PBC), which needs to be established between the producer information system (active system) and the digital preservation system (DPS). The PBC is an XML-file containing specifications and regulations in the form of structured information and rules that upholds the agreed-on structure and content of information packages and execution paths in the PoF middleware. The CaPM-PBC is always part of the execution of a preservation preparation workflow defined in D8.5 [Gallo et al., 2016] by receiving and dispatching requests from other PoF middleware components and by interacting with the PoF workflow infrastructure. A typical example of such interaction is provided by the Collector component: when retrieving objects from the active system it will always interact with the CaPM-PBC to identify from which active system it will fetch the objects by the retrieval of connection end-point information, if there are instructions regarding the fetch, such as if physical/logical structure should be included, and if fetched objects is according to expectations regarding anticipated mime types, and that limit values for a single fetch is not exceeded. Another example is when the Archiver component creates the submission information package: the Archiver retrieves information about metadata standards to use. The Archiver also retrieves information from the PBC about the preservation organization, contact information, systems etc. used as provenance metadata. It also retrieves information about the package folder structure, definition of fixity algorithm, and the connection endpoint of the preservation system. Another example is related to the re-activation of content archived in the Preservation System, the CaPM-PBC provides information such as if there should be a migration at access and the migration path for a specific mime-type.

Besides being part of the preservation preparation and re-activation workflows the CaPM-PBC supports the scheduled transformation mechanism executed by the Preservation Aware Storage System [Chen et al., 2016] by providing agreed rules on the management of original objects and any copies. The CAPM-PBC may also contain information about agreed management of copies in the preservation system, as well as mechanisms regarding integrity checks, and how events in the preservation system should be communicated back to the active system. This information is not intended to be executed “on-the-fly” in the preservation system, instead used as basis for manual configuration of mechanisms in the preservation system. The CaPM-PBC is invoked through a REST API based on the request path; different information from a specific PBC, identified by its ID, is retrieved. Another responsibility for the CaPM component is to support monitoring of DP activities (Activity Logging) executed by other PoF middleware components. A typical example for the use of the CaPM-AL is to check status of the execution of activities in the preservation preparation workflow; the Archiver checks the logs created by the CaPM-AL to ensure that activities that should be executed according to the PBC has been carried out without errors. If an error is detected it sends a message to the PoF workflow manager. A REST API could also be invoked to request information from the CaPM-AL. Due to it's location in the middle of the interaction between active systems and preservation systems, the CaPM is also able to keep track of every object that passes through the PoF middleware. This functionality is referred to as Preservation Planning Support (PPS) and provides a bi-directional identification of systems involved, identified by the PBC, and the identification and logging of mime type versions that are part of an interaction. This information is useful as input to different preservation planning scenarios for detection of file format obsolescence and decision support in the choice of target file format in a migration scenario. This information is available by a REST API as a JSON output which can be imported by any preservation system and also provided via a web GUI through the CaPM-PPS.

WP and Deliverables The CaPM is developed within WP5 (Joint Information and Preservation Management). This is the first release of the component that was previously described in deliverable D5.3 [Nilsson et al., 2015], the contributing partners are mainly LTU, EURIX, IBM, DFKI, and dkd.

API and I/O Formats The CaPM component utilises common APIs as described under section 5.2 with the following additions. The web-GUI provided by the CaPM-PPS is implemented by the use of PrimeFaces[14] and JSF API[15] and the DB communication uses Java Persistence API. The classes that constitute the CaPM component are buillt to the `CaPM.jar` software package and the `capmmimeviewer.war`

Status and Workplan The current solution for the Context-aware Preservation Manager provides all functionalities expected within the project. The integration of it still needs improvements; which for example could mean different execution paths in the (PoF) Middleware based on information from the Preservation Broker Contract.

[14] http://www.primefaces.org/
[15] http://www.oracle.com/technetwork/java/javaee/javaserverfaces-139869.html

License The source code of this component is released as Open Source, as part of the PoF Middleware code.

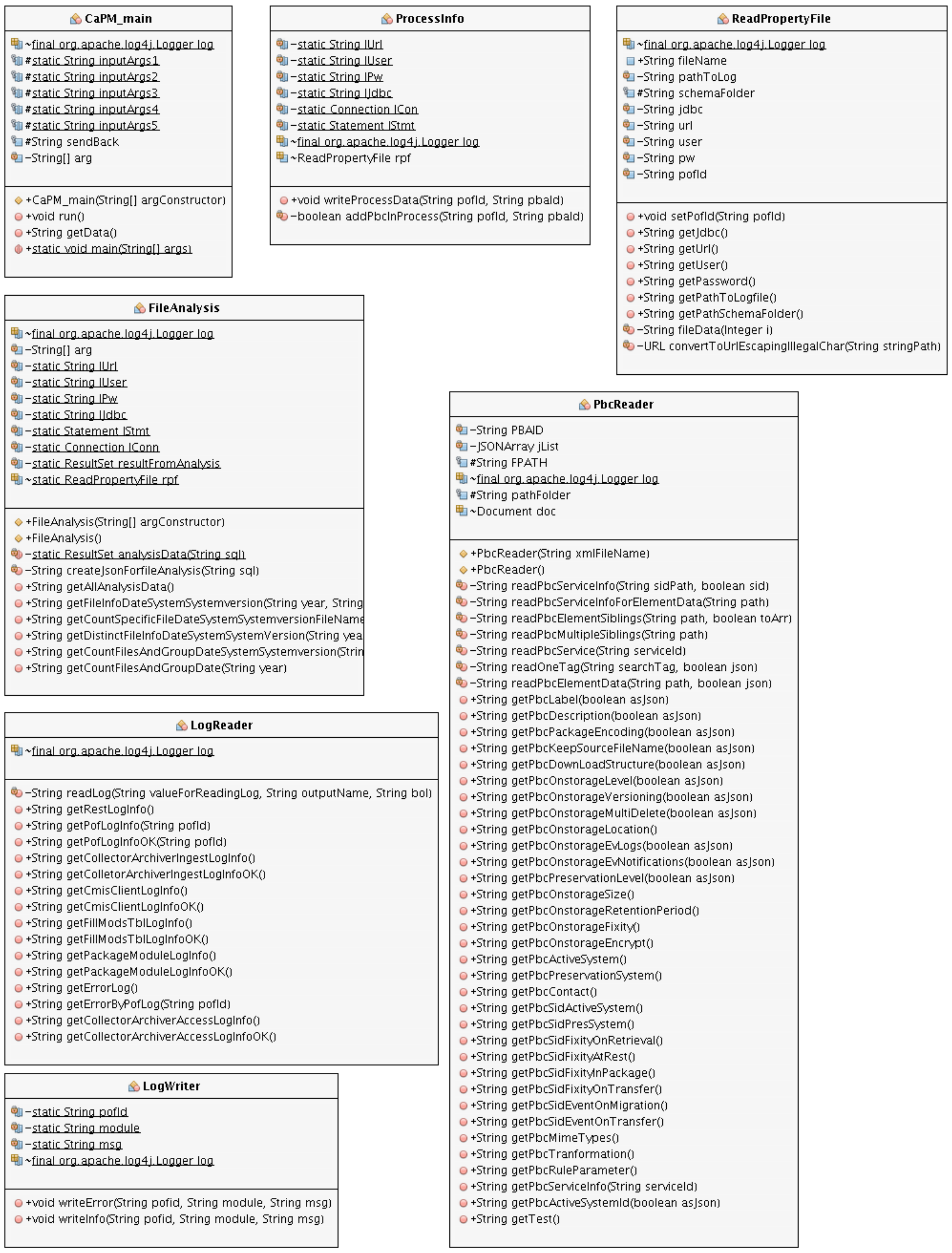

CaPM_main
~final org.apache.log4j.Logger log
#static String inputArgs1
#static String inputArgs2
#static String inputArgs3
#static String inputArgs4
#static String inputArgs5
#String sendBack
−String[] arg
+CaPM_main(String[] argConstructor)
+void run()
+String getData()
+static void main(String[] args)
ProcessInfo
−static String lUrl
−static String lUser
−static String lPw
−static String lJdbc
−static Connection lCon
−static Statement lStmt
~final org.apache.log4j.Logger log
~ReadPropertyFile rpf
+void writeProcessData(String pofId, String pbaId)
−boolean addPbcInProcess(String pofId, String pbaId)
ReadPropertyFile
~final org.apache.log4j.Logger log
+String fileName
−String pathToLog
#String schemaFolder
−String jdbc
−String url
−String user
−String pw
−String pofId
+void setPofId(String pofId)
+String getJdbc()
+String getUrl()
+String getUser()
+String getPassword()
+String getPathToLogfile()
+String getPathSchemaFolder()
−String fileData(Integer i)
−URL convertToUrlEscapingIllegalChar(String stringPath)
FileAnalysis
~final org.apache.log4j.Logger log
−String[] arg
−static String lUrl
−static String lUser
−static String lPw
−static String lJdbc
−static Statement lStmt
−static Connection lConn
−static ResultSet resultFromAnalysis
~static ReadPropertyFile rpf
+FileAnalysis(String[] argConstructor)
+FileAnalysis()
−static ResultSet analysisData(String sql)
−String createJsonForfileAnalysis(String sql)
+String getAllAnalysisData()
+String getFileInfoDateSystemSystemversion(String year, String
+String getCountSpecificFileDateSystemSystemversionFileName
+String getDistinctFileInfoDateSystemSystemVersion(String yea
+String getCountFilesAndGroupDateSystemSystemversion(Strin
+String getCountFilesAndGroupDate(String year)
PbcReader
−String PBAID
−JSONArray jList
#String FPATH
~final org.apache.log4j.Logger log
#String pathFolder
~Document doc
+PbcReader(String xmlFileName)
+PbcReader()
−String readPbcServiceInfo(String sidPath, boolean sid)
−String readPbcServiceInfoForElementData(String path)
−String readPbcElementSiblings(String path, boolean toArr)
−String readPbcMultipleSiblings(String path)
−String readPbcService(String serviceId)
−String readOneTag(String searchTag, boolean json)
−String readPbcElementData(String path, boolean json)
+String getPbcLabel(boolean asJson)
+String getPbcDescription(boolean asJson)
+String getPbcPackageEncoding(boolean asJson)
+String getPbcKeepSourceFileName(boolean asJson)
+String getPbcDownLoadStructure(boolean asJson)
+String getPbcOnstorageLevel(boolean asJson)
+String getPbcOnstorageVersioning(boolean asJson)
+String getPbcOnstorageMultiDelete(boolean asJson)
+String getPbcOnstorageLocation()
+String getPbcOnstorageEvLogs(boolean asJson)
+String getPbcOnstorageEvNotifications(boolean asJson)
+String getPbcPreservationLevel(boolean asJson)
+String getPbcOnstorageSize()
+String getPbcOnstorageRetentionPeriod()
+String getPbcOnstorageFixity()
+String getPbcOnstorageEncrypt()
+String getPbcActiveSystem()
+String getPbcPreservationSystem()
+String getPbcContact()
+String getPbcSidActiveSystem()
+String getPbcSidPresSystem()
+String getPbcSidFixityOnRetrieval()
+String getPbcSidFixityAtRest()
+String getPbcSidFixityInPackage()
+String getPbcSidFixityOnTransfer()
+String getPbcSidEventOnMigration()
+String getPbcSidEventOnTransfer()
+String getPbcMimeTypes()
+String getPbcTranformation()
+String getPbcRuleParameter()
+String getPbcServiceInfo(String serviceId)
+String getPbcActiveSystemId(boolean asJson)
+String getTest()
LogReader
~final org.apache.log4j.Logger log
−String readLog(String valueForReadingLog, String outputName, String bol)
+String getRestLogInfo()
+String getPofLogInfo(String pofId)
+String getPofLogInfoOK(String pofId)
+String getCollectorArchiverIngestLogInfo()
+String getColletorArchiverIngestLogInfoOK()
+String getCmisClientLogInfo()
+String getCmisClientLogInfoOK()
+String getFillModsTblLogInfo()
+String getFillModsTblLogInfoOK()
+String getPackageModuleLogInfo()
+String getPackageModuleLogInfoOK()
+String getErrorLog()
+String getErrorByPofLog(String pofId)
+String getCollectorArchiverAccessLogInfo()
+String getCollectorArchiverAccessLogInfoOK()
LogWriter
−static String pofId
−static String module
−static String msg
~final org.apache.log4j.Logger log
+void writeError(String pofid, String module, String msg)
+void writeInfo(String pofid, String module, String msg)

Figure 11: Class diagram for Context-aware Preservation Manager component

5.7.1 CaPM - Preservation Broker Contract (CaPM-PBC)

This component is responsible for the management of agreements that provide structure to the relationships and interactions between Active Systems and DPS. The information that is managed by the CaPM-PBC is defined in an XML-schema and based on this schema a default PBC (XML) is generated that reflects the agreement between the producer (active system) and the archive (preservation system). A PBC contains structured information divided into the different sections of identification, collectionType, services, actions, and rules as depicted in Figure 12. The information from each section of the PBC is available through a REST API, which makes it available from any clients that supports REST and a JSON parser. A full-fledged version of the PBC-schema and an example of a generated PBC-file is accessible in the appendix.

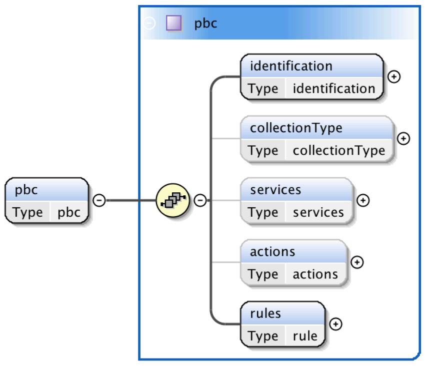


Figure 12: CaPM - Preservation Broker Contract (PBC)

5.7.1.1 CaPM-PBC-identification

The identification section (Figure 13) contains information about the PBC, such as identification of the contract, preservation level which identifies the template from which the contract is generated, a preservation value that defines the threshold level for appraisal, and information active system and preservation system participated in the interaction defined by the contract. Information from this PBC section is accessible through a REST API described in detail in 5.7.1.3

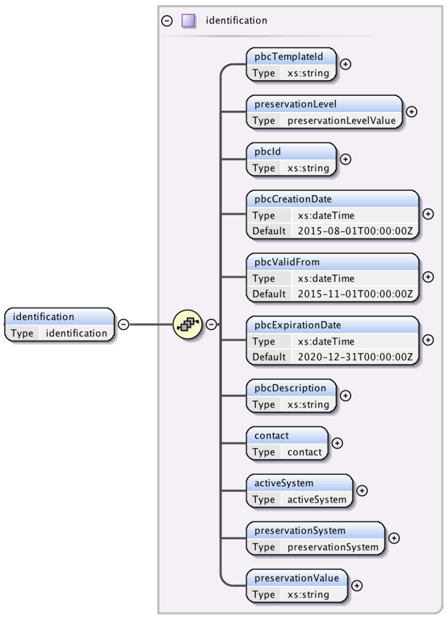


Figure 13: CaPM - PBC - identification section

5.7.1.2 CaPM-RESTful services for the Preservation Broker Contract (PBC)

REST APIs are published using Jersey, the reference implementation of JAX-RS specification for RESTful web services. In the following we list the available APIs with the expected parameters and the output format. Table 5 shows the common REST APIs for the CaPM-PBC component.

Server path	/capmrest/webresources/capmservice
Supported response types	JSON
HTTP request type	GET
Description	CaPM-PBC REST API: returns information from specified sections in a Preservation Broker Contract (PBC)

Table 5: CaPM-PBC REST common interface

5.7.1.3 CaPM–PBC–REST-identification section

Table 6 shows a subset of RESTful services exposing information from the CaPM-PBC identification section. There exists similar services for exposing every element from the contract, but those described here cover the most relevant information.

Request path	/preservationsystem/{pbc-1}
Response	`{"serviceId":"sid-2","location":"Italy","name":"EURIX DSpace", "ownedBy":"EURIX Group","Id":"ps-123","version":"1.0.1"}`
Description	**Parameters:** preservationsystem=path to internal method, pbc-1= identification of the preservation broker contract (PBC). **Returns information**: about the preservation system related to the PBC. **Example of use:** by the CaPM-PPS component that logs content information related to a preservation system part of PoF workflows.
Request path	/activesysteminfo/{pbc-1}
Response	`{"serviceId":"sid-1","name":"EURIXGROUP", "ownedBy":"EURIXGROUP","Id":"eurix-mw-1", "type":"personal information from middleware","version":"1.0"}`
Description	**Parameters:** activesysteminfo=path to internal method, pbc-1= identification of the preservation broker contract (PBC). **Returns information:** about the producer information system (active system) part of an interaction. **Example of use:** this is used by the Archiver component as metadata in SIP and by the CaPM-PPS component as part of statistical data that relates objects to specific information systems.
Request path	/contact/{pbc-1}
Response	`{"ownerName":"Goran Lindqvist",,"ownerEmail":"goran@ltu.se", "alternativeContactName":"Ingemar Andersson", "alternativeContactEmail":"ingo@ltu.se}`
Description	**Parameters:** contact=path to internal method, pbc-1= identification of the preservation broker contract (PBC). **Returns information:** personal contact information about persons responsible for transfer of objects from producer side (active system). **Example of use:** by the Archiver component as subset of provenance metadata in SIP.
Request path	/preservationlevel/{pbc-1}
Response	`{"preservationLevel":"premium"}`
Description	**Parameters:** preservationlevel=path to internal method, pbc-1= identification of the preservation broker contract (PBC). **Returns information:** that specifies the PBC base configuration (template) selected by the information producer (users of active system). **Example of use:** defines a name for different types of PBC that reflects different configurations as premium, standard, or basic. Different preservation service providers could offer different types of PBC templates. The purpose is to facilitate a simplified PBC configuration process for novice users.

Table 6: CaPM-PBC-REST-Identification

5.7.1.4 CaPM-PBC-CollectionType section

The collectionType section, depicted in Figure 14, of the PBC includes information about size and quantity limits, list of expected file formats, stating if source file name should be preserved, and a list of transformation rules defining the event trigger, source and target file format for each migration action. The information in this PBC-section is accessible by a REST API described in detail in

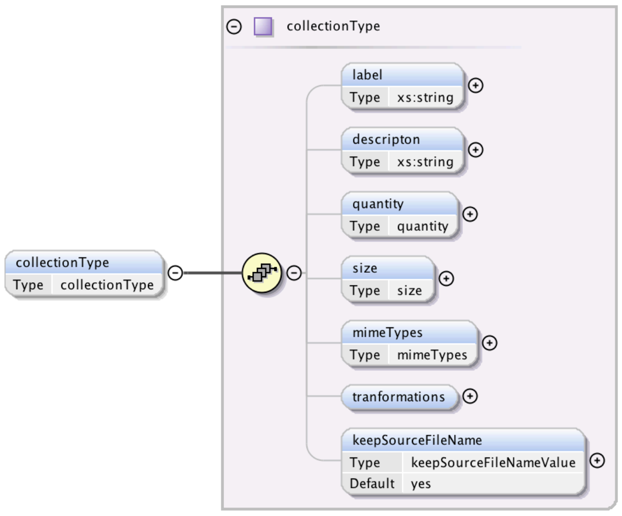


Figure 14: CaPM - PBC - collectionType section

5.7.1.5 CaPM–PBC–REST-collectionType section

Table 7 shows a subset of RESTful services exposing information from the CaPM-PBC collectionType section. There exists similar services for exposing every element from the contract, but those described here show the more elaborate ones.

Request path	/mimetypes/{pbc-1}
Response	`[{"mimeType":"application\/pdf"}, {"mimeType":"image\/jpeg"}, {"mimeType":"image\/png"}, {"mimeType":"image\/bmp"}, {"mimeType":"image\/tiff"}]`
Description	**Parameters:** mimetypes=path to internal method, pbc-1= identification of the preservation broker contract (PBC). **Returns information:** about "acceptable" mime types defined as an agreement between information producer (active system) and archive (preservation system). **Example of use:** by the Collector component comparing list of acceptable mime types for objects on pending request to be fetched from active system.
Request path	/transformation/{pbc-1}
Response	`[{"onAction":"ingest","origPuid":"fmt\/22", "keepOriginal":"yes","origMimeType":"image\/png", "serviceId":"sid-3","targetPuid":"fmt\/11", "targetMimeType":"image\/tiff"}, {"onAction":"access","origPuid":"fmt\/22", "keepOriginal":"yes","origMimeType":"image\/bmp", "serviceId":"String","targetPuid":"fmt\/11", "targetMimeType":"image\/jpeg"}]`
Description	**Parameters:** transformation=path to internal method, pbc-1= identification of the preservation broker contract (PBC). **Returns information:** rules for automated migration actions containing information defining which objects should be migrated to specified file formats identified by its mime type. This information also states the event when migration action should be executed (ingest/access). **Example of use:** could be used in migration scenario executed in the PoF middleware as part of preservation preparation or re-activation workflow. It is also possible to use this as basis for configuration of migration actions executed at preservation system. The use of this rule is not supported by the PoF middleware in this release.
Request path	/keepsourcefilename/{pbc-1}
Response	`{"keepSourceFileName":"true"}`
Description	**Parameters:** keepsourcefilename=path to internal method, pbc-1= identification of the preservation broker contract (PBC). **Returns information:** that defines if source file name and path should be kept as provenance metadata. **Example of use:** as input to the Archiver component as a rule defining how to manage source data and if it should be added as metadata in SIP. This information could also be used as a rule at preservation system side when creating an Archival Information Package (AIP).
Request path	/collectionlabel/{pbc-1}
Response	`{"label":"private photos"}`
Description	**Parameters:** collectionlabel=path to internal method, pbc-1= identification of the preservation broker contract (PBC). **Returns information:** categorisation metadata. **Example of use:** the Archiver components use this as metadata when creating a SIP. It could also be used in "collection" scenarios to keep track of objects fetched at different occasions.

Table 7: CaPM-PBC-CollectionType

5.7.1.6 CaPM-PBC-services section

The services section, depicted in Figure 15, of the PBC provides information about tools and service endpoints that defines how to communicate with a service. The information in this section is used by different components in the PoF middleware either during execution of PoF workflows or as static information used as the basis for component configuration and documentation of an agreement between active system and preservation system. The information in this PBC-section is accessible by a REST API described in detail in 5.7.1.7.

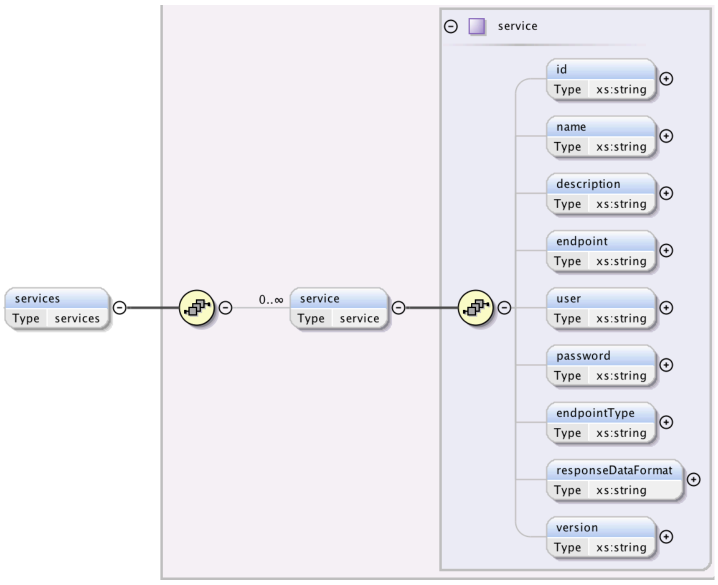


Figure 15: CaPM - PBC - services section

5.7.1.7 CaPM-PBC-REST-services section

Table 8 shows a subset of RESTful services exposing information from the CaPM-PBC services section. The selection was made to show the breadth of services.

Request path	/sidactivesystem/{pbc-1}
Response	`{"endpointType":"CMIS",` `"description":"CMIS connection to PIMO CMIS endpoint",` `"name":"stainer","responseDataFormat":"XML","password":"stainerstainer",` `"user":"stainer",` `"endpoint":"http:\/\/192.168.253.8:8080\/cmis\/atom11","version":"1.0"}`
Description	**Parameters:** sidactivesystem=path to internal method, pbc-1= identification of the preservation broker contract (PBC). **Returns information:** about the connection endpoint to active system. **Example of use:** by the Collector component at execution as connection information to a CMIS-endpoint for fetching of content from active system.
Request path	/sidpreservationsystem/{pbc-3}
Response	`{"endpointType":"REST_JAX_RS",` `"description":"DSpace repository for ForgetIT project",` `"name":"EURIX repository endpoint","responseDataFormat":"JSON",` `"password":"String","user":"String",` `"endpoint":"http:\/\/archive\/oais-api\/restws\/ingest\/sip",` `"version":"1.1"}`
Description	**Parameters:** sidpreservationsystem=path to internal method, pbc-1= identification of the preservation broker contract (PBC). **Returns information:** about the connection endpoint to preservation system. **Example of use:** by the Archiver component at execution as connection information when connecting to a preservation system for transfer of Submission Information Package (SIP).
Request path	/serviceinfo/{pbc-1}
Response	`{"endpointType":"Storlet",` `"description":"JPEG File Interchange Format to PNG",` `"name":"Stellent Image Export","responseDataFormat":"JSON",` `"password":"String",` `"user":"String",` `"endpoint":"Preservation-aware Storage Services","version":"1.01"}`
Description	**Parameters:** serviceinfo=path to internal method, pbc-1= identification of the preservation broker contract (PBC). **Returns information:** about a service endpoint that is part of the interaction between an active system and preservation system executed somewhere in a defined PoF workflow. **Example of use:** the Collector/Archiver components use this information when establishing a communication endpoint to an active system or preservation system. It could also define other services/tools/algorithms that is used by a component in a PoF middleware workflow.

Table 8: CaPM-PBC-REST-Services

5.7.1.8 *CaPM-PBC-actions section*

The actions section, depicted in Figure 16, of the PBC provides information about tools and service endpoints that defines how to communicate with a service. The information in this section is used by different components in the PoF middleware either at execution in PoF workflows or as static information as basis for component configuration and documentation of an agreement between active system and preservation system. The information in this PBC-section is accessible by a REST API described in detail in 5.7.1.9.

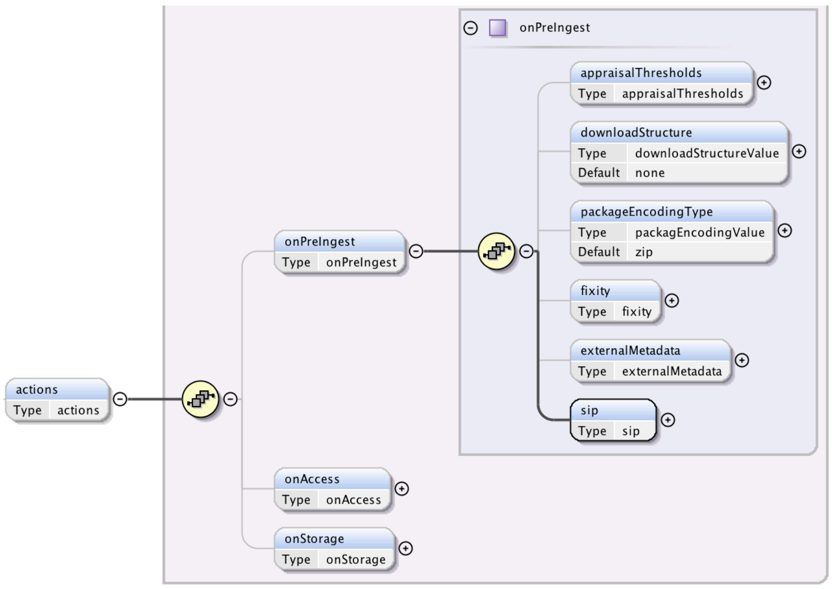


Figure 16: CaPM - PBC - actions section

5.7.1.9 *CaPM-PBC-REST-actions section*

Table 9 shows a subset of RESTful services exposing information from the CaPM-PBC actions section. The selection shows a variety of the services that expose the contract information.

Request path	/downloadstructure/{pbc-1}
Response	`{"downloadStructure":"true"}`
Description	**Parameters:** downloadstructure=path to internal method, pbc-1= identification of the preservation broker contract (PBC). **Returns information:** that defines if “structure” information should be part of data fetched from active system. **Example of use:** by the Collector component as configuration of CMIS client when fetching objects from active system.
Request path	/packageencoding/{pbc-1}
Response	`{"packageEncodingType":"zip"}`
Description	**Parameters:** packageencoding=path to internal method, pbc-1= identification of the preservation broker contract (PBC). **Returns information:** that specifies preferred package encoding (zip, tar etc.). **Example of use:** by the Archiver component when creating a SIP adjusted to receiving preservation system.
Request path	/onstoragefixity/{pbc-1}
Response	`{"Interval":"24","intervalUnit":"month","algorithm":"md5"}`
Description	**Returns information:** about an integrity protection rule that reflects a preservation policy. **Example of use:** this is not part of any PoF workflow; intended for use as a stated agreement having impact on a preservation system configuration.
Request path	/onstorageencrypt/{pbc-1}
Response	`{"keyAtTransfer":"SSL","keyAtRest":"SSE"}`
Description	**Parameters:** onstorageencrypt=path to internal method, pbc-1= identification of the preservation broker contract (PBC). **Returns information:** about a rule that reflects a security policy. **Example of use:** as part of object management in PoF middleware defining need of encryption mechanisms at transfer and storage. This should also have an impact on preservation system configuration. The use of this rule is not supported by the PoF middleware in this release.
Request path	/onstoragelevel/{pbc-1}
Response	`{"level":"silver"}`
Description	**Parameters:** onstoragelevel=path to internal method, pbc-1= identification of the preservation broker contract (PBC). **Returns information:** about a rule that defines management of object copies at the preservation system side. This value could e.g. be defined as premium, gold, silver, or bronze. **Example of use:** as input to storage configuration where each level could be defined as for platinum; at least three copies at different locations with different disaster threats to bronze level stating; two copies that is not collocated.
Request path	/onstoragelocation/{pbc-1}
Response	`{"location":"EU"}`
Description	**Parameters:** onstoragelocation=path to internal method, pbc-1= identification of the preservation broker contract (PBC). **Returns information:** reflecting a preservation policy defining restrictions on storage location. **Example of use:** as input to storage configuration at preservation system side reflecting legal restrictions on storage location.
Request path	/onstoragemultidelete/{pbc-1}
Response	`{"multiFactorDelete":"yes"}`
Description	**Parameters:** onstoragemultidelete=path to internal method, pbc-1= identification of the preservation broker contract (PBC). **Returns information:** reflecting a preservation policy defining restrictions on deletes action. **Example of use:** as input to storage configuration at preservation system side reflecting legal restrictions on object management on order to prevent accidental deletion of objects.
Request path	/onstorageretentionperiod/{pbc-1}
Response	`{"min":"0","endOfRetentionAction":"0","max":"0","unitType":"GB", "nextLocation":"none","nextLevel":"bronze"}`
Description	**Parameters:** onstorageretentionperiod=path to internal method, pbc-1= identification of the preservation broker contract (PBC). **Returns information:** reflecting a preservation policy defining rules on lifecycle management of objects at preservation system side. **Example of use:** as input to storage configuration that could trigger transfer of objects to low-cost storage or scheduled deletions.
Request path	/onstorageversioning/{pbc-1}
Response	`{"versioning":"2"}`
Description	**Parameters:** onstorageversioning=path to internal method, pbc-1= identification of the preservation broker contract (PBC). **Returns information:** that defines a rule specifying number of versions that needs to be managed. **Example of use:** as input to preservation system, exceeding this number could mean that older versions could be discarded. Not part of scenario executed in the ForgetIT project.
Request path	/onstoragesize /{pbc-1}
Response	`{"max":"0","unitType":"GB"}`
Description	**Parameters:** onstoragesize=path to internal method, pbc-1= identification of the preservation broker contract (PBC). **Returns information:** about agreed maximum storage volume as a trigger for the need of a review of stated agreement. **Example of use:** as a way to limit the PBC validity.

Table 9: CaPM – PBC-REST-Actions

5.7.1.10 CaPM-PBC-rules section

The rules section, depicted in Figure 17, of the PBC provides the ability to specify various rules and its bundled actions, reflecting preservation policies that can be applied for different activities in a PoF workflow. This information could be applied either at execution time or used as static information as the basis for configuration of a PoF component activity. The information in this PBC-section is accessible by a REST API described in detail in 5.7.1.11.

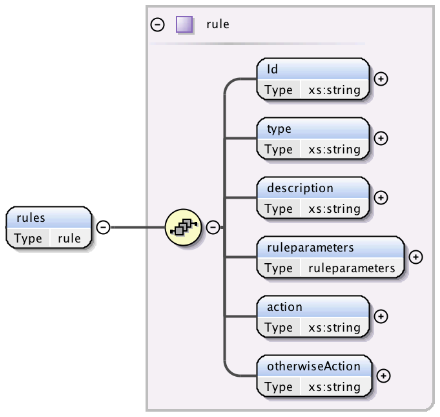


Figure 17: CaPM - PBC - rules section

5.7.1.11 CaPM-PBC-REST-rules section

Table 10 shows one example of a RESTful service exposing information from the CaPM-PBC rules section. Rules should be constructed in similar ways, and thereby this service should work well for many different cases.

Request path	/ruleparameter/{pbc-3}

Response	`[{"dependency":"none","condition":"equals","name":"PL1","value":"premium"},`
	`{"dependency":"PL1","condition":"aboveEquals","name":"QT1","value":"0.95"},`
	`{"dependency":"QT1","condition":"equals","name":"mimetype","value":"default"},`
	`{"dependency":"PL1","condition":"aboveEquals","name":"QT2","value":"0.92"},`
	`{"dependency":"QT2","condition":"equals","name":"mimetype","value":"image\/jpg"]`

Description	**Returns information:** about “rules”, agreed upon between information producer and archive, on how to manage objects and its copies in transformation actions at storage level. The response contains information relating Quality Thresholds (QT) for different mime types. There could be different “rules” related to different Preservation Levels (PL). The value of QT is used in comparison to the result returned from the quality of a migrated object. If the QT is reached a related action (not part of the example response) is expressing if the original object could be deleted or not and how to handle copies. The transformation and quality assessment actions are executed by storlets close to data in the Preservation Aware Storage Engine. **Example of use:** the Preservation Aware Storage System uses this in scheduled transformation actions as rules executed in the Storlet Engine component presented in D7.4 [Chen et al., 2016].

Table 10: CaPM-PBC-Rules

5.7.2 CaPM - Activity Logger (CaPM-AL)

The Activity Logger is the functional entity, part of the Context-aware Preservation Manager (CaPM) component, that is responsible for logging of actions executed by PoF Middleware components. This entity makes it possible to monitor activities in the PoF middleware and to take actions depended on the result of a functional execution in a middleware component. The features of this component make it possible to execute an alternative workflow depending on the status of a component execution. For example, there is a possibility to check logged entries with error status containing information that identifies the function causing the problem. Another example is the use of threading mechanisms in the PoF middleware that makes it difficult to know when a specific function has finished its execution; therefore, the logging by the CaPM-AL could be used to keep track of execution status in each PoF workflow. The information created by the CaPM-AL can be requested by a REST API, which makes it available from any clients that supports REST and a JSON parser as described in detail in 5.7.2.1. The logging mechanism in CaPM-AL is based on the use of Apache log4j and part of the logged data is fetched from the CaPM-PBC and the output during processing is stored in log4j-application.log file and certain output data is stored permanently in DB.

5.7.2.1 CaPM-AL-RESTful-API

Table 11 shows a subset of RESTful services exposing information from the CaPM-AL. These services use the same base service path as defined in Table 5.

Request path	/collectorarchiveringestloginfo/{abx-221-a}
Response	`{"CollectorArchiveIngest module":["2015-11-30 07:43:10 INFO Collector:83 - [CollectorArchiverIngest:Collector:run] started","2015-11-30 07:43:10 INFO Collector:215 - CollectorArchiverIngest:Collector:run, OnetestIng-mw231 Start downloading file from CMIS server","2015-11-30 07:43:10 INFO Collector:529 – CollectorArchiverIngest:Collector:createFolderOnHD ...etc`
Description	**Returns information:** from the log4j-application.log file filtered on logged entries with INFO as status and from a specified functional entity identified by its request path. This information is logged as a tracker for process status and could be executed inside a PoF component. Every entry is identified by a UUID (PoF-Id) that is generated for each workflow in the PoF middleware. The PoF-Id is linked to a PBC-Id and logged in a DB handled by the ProcessInfo class as depicted in the CaPM class diagram in Figure 13. **Example of use:** the Collector and Archiver components have implemented support for this in every functional entity and use this for logging of process status. The response shows a subset from the log added by the Collector at start of fetching content from an active system.
Request path	/errorlog/{abx-221-a}
Response	`{"Error":["2015-11-30 09:12:45 ERROR PbcReader:54 - CaPM:PbcReader():error..: \/opt\/forgetit\/run\/schema\/Value1.xml (No such file or directory)","2015-12-01 08:16:09 ERROR LogWriter:31 – abx-221-a: ...etc`
Description	**Returns information:** from the log4j-application.log file filtered on logged entries with ERROR as status. Every entry is identified by a UUID (PoF-Id) that is generated for each workflow in the PoF middleware. The PoF-Id is linked to a PBC-Id and logged in a DB handled by the ProcessInfo class as depicted in the CaPM class diagram in Figure 13. **Example of use:** the Collector and Archiver components have implemented support for this in every functional entity that has a catch of exceptions. The Archiver uses this for detection of any errors before start of the creation of a submission information package (SIP). If an error is detected an alternative workflow of actions is executed depended on error code.

Table 11: CaPM-Activity Logger-REST-API

5.7.3 CaPM – Preservation Planning Support (CaPM-PPS)

It might be argued that the functionality supported by the Preservation Planning Support (PPS) is a natural consequence of CaPMs location as a broker between active information systems and preservation systems, and this is true; there is no better location for a component with the responsibility to monitor evolution of technology (technology watch) as depicted in Figure 7 "Setting Change Workflow". The PPS identifies and logs every single object that is passing through the PoF middleware in a bi-directional communication, executed in the preservation preparation workflow as depicted in Figure 3 and re-activation workflow as depicted in Figure 5. Besides the logging of objects it also keep track of the systems involved in the PoF workflows. With this information as a basis the PPS is able to support different scenarios; a preservation system could set up technology watch triggers for different thresholds based on the use of file format; what is requested for re-use by active systems and what is appraised for preservation. A preservation system could also make a request for usage statistics for a specific file format to be used as a complement to its internal preservation-planning component. The data held by the CaPM-PPS could also be used as preparation of input for a re-activation scenario (Figure 5); it could state that a specific version of a system is able to use a specific range of file formats as a basis for a format transformation specification registered in a PBC. Every object is identified by the use of DROID[16], which generates a PRONOM Unique Identifier (PUID)[17] for each identified file format. This makes it possible to link the data generated by this component to the PRONOM[18] technical registry held by The National Archives. The PRONOM registry provides information about an identified file

[16] https://github.com/digital-preservation/droid
[17] http://www.nationalarchives.gov.uk/aboutapps/pronom/puid.htm
[18] https://www.nationalarchives.gov.uk/PRONOM/

format containing links to external tools and services useful in preservation planning (migration pathway planning) scenarios. The information from created by the CaPM-PPS could be requested by a REST API as described in detail in 5.7.3.1, which makes it available from any clients that supports REST and a JSON parser. The CaPM-PPS also provides a Web based User Interface, depicted in Figure 18 that exposes a summarization of identified file formats that has been managed by the PoF middleware. The first version of the Web GUI provides the possibility to filter on active systems and mime type classifications. The class diagram depicted in Figure 19 shows the `Fileinfo` class that's managing the information about files and systems involved in PoF workflows, the other classes in Figure 19 is helper classes for visualizing the data as shown in Figure 18.

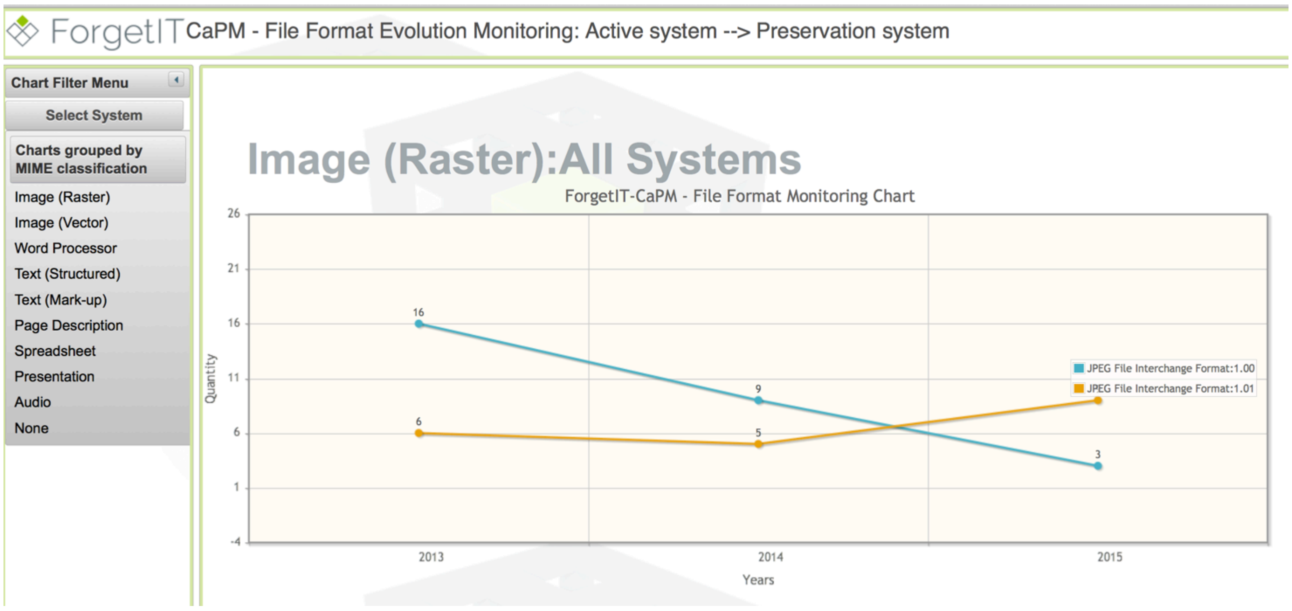

ForgetIT CaPM - File Format Evolution Monitoring: Active system --> Preservation system
Chart Filter Menu
Select System
Charts grouped by MIME classification
Image (Raster)
Image (Vector)
Word Processor
Text (Structured)
Text (Mark-up)
Page Description
Spreadsheet
Presentation
Audio
None
Image (Raster):All Systems
ForgetIT-CaPM - File Format Monitoring Chart
Quantity
Years
2013
2014
2015
JPEG File Interchange Format:1.00
JPEG File Interchange Format:1.01
16
9
3
6
5

Figure 18: Web interface of the CaPM-Preservation Planning Support (CaPM-PPS), that shows summarized values on number of objects that has been managed by the PoF middleware grouped on file format version per year. It is possible to filter on active system and file format classification.

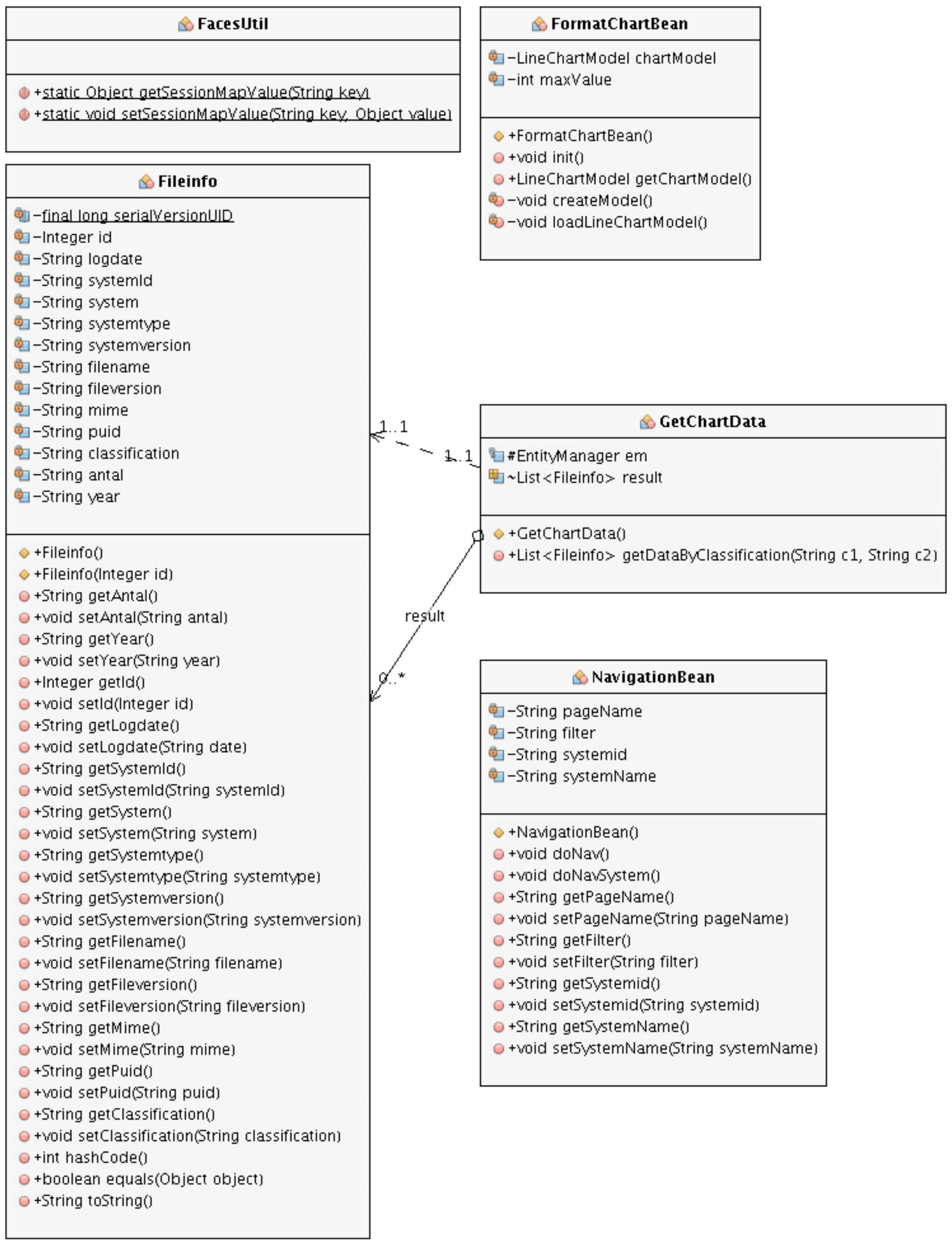

FacesUtil
+static Object getSessionMapValue(String key)
+static void setSessionMapValue(String key, Object value)
FormatChartBean
−LineChartModel chartModel
−int maxValue
+FormatChartBean()
+void init()
+LineChartModel getChartModel()
−void createModel()
−void loadLineChartModel()
Fileinfo
−final long serialVersionUID
−Integer id
−String logdate
−String systemId
−String system
−String systemtype
−String systemversion
−String filename
−String fileversion
−String mime
−String puid
−String classification
−String antal
−String year
+Fileinfo()
+Fileinfo(Integer id)
+String getAntal()
+void setAntal(String antal)
+String getYear()
+void setYear(String year)
+Integer getId()
+void setId(Integer id)
+String getLogdate()
+void setLogdate(String date)
+String getSystemId()
+void setSystemId(String systemId)
+String getSystem()
+void setSystem(String system)
+String getSystemtype()
+void setSystemtype(String systemtype)
+String getSystemversion()
+void setSystemversion(String systemversion)
+String getFilename()
+void setFilename(String filename)
+String getFileversion()
+void setFileversion(String fileversion)
+String getMime()
+void setMime(String mime)
+String getPuid()
+void setPuid(String puid)
+String getClassification()
+void setClassification(String classification)
+int hashCode()
+boolean equals(Object object)
+String toString()
1..1
1..1
GetChartData
#EntityManager em
~List<Fileinfo> result
+GetChartData()
+List<Fileinfo> getDataByClassification(String c1, String c2)
result
0..*
NavigationBean
−String pageName
−String filter
−String systemid
−String systemName
+NavigationBean()
+void doNav()
+void doNavSystem()
+String getPageName()
+void setPageName(String pageName)
+String getFilter()
+void setFilter(String filter)
+String getSystemid()
+void setSystemid(String systemid)
+String getSystemName()
+void setSystemName(String systemName)

Figure 19: Class diagram Context-aware Preservation Manager component – Preservation Planning Support – Web-GUI

5.7.3.1 CaPM-PPS-REST-API

Table 12 shows a subset of RESTful services exposing information from the CaPM-PPS. These services use the same base service path as defined Table 5 and were selected to show possibilities of the PPS.

Request path	/countfilesandgroupdate
Response	`[{"logdate":"2016-01-11","filename":"Acrobat PDF 1.2 - Portable Document Format","nmbr":"1","fileversion":"1.2"}, {"logdate":"2016-01-11","filename":"Extensible Markup Language","nmbr":"1","fileversion":"1.0"}, {"logdate":"2016-01-11","filename":"JPEG File Interchange Format","nmbr":"1","fileversion":"1.02"}, {"logdate":"2016-01-11","filename":"MPEG-1 Program Stream","nmbr":"1","fileversion":"none"},{"logdate":"2016-01-11","filename":"Windows Bitmap","nmbr":"1","fileversion":"3.0"}]`
Description	**Returns information:** from log data that keep track of every object passing the PoF middleware containing information about the log date, file format, version, and number of files. **Example of use:** as statistical use of file format data to any preservation planning functional entity in an OAIS compatible preservation system.
Request path	/fileinfodatesystemversion/{2015}/{EURIX}/{1.2}
Response	`[{"processType":"ingest","logdate":"2016-01-11","id":2,"system":"EURIX content management","pbaId":"pbc-2", "filename":"Acrobat PDF 1.2 - Portable Document Format", "systemtype":"Content Management System", "mime":"application\/pdf","systemversion":"1.2","systemId":"eurix-mw-1","puid":"fmt\/16","fileversion":"1.2"},{"processType":"ingest"," etc.`
Description	**Returns information:** about log data that keep track of every object passing the PoF middleware containing information about the file format, version, mime type, log date, active system, PBC-Id, the pronom unique identifier (puid), and the event (ingest/access) filtered on year, and version of an active system. **Example of use:** as statistical use of file format data to preservation planning that need to be interconnected to type of system. This data could be used as input to preparation for a re-activation scenario.

Table 12: CaPM-Preservation Planning Support-REST-API

6 Summary

As indicated by the success/progress indicators discussion in sections 3 and 4, the transition of objects between active systems and preservation systems are in good shape. With the introduction of the Context-aware Preservation Manager, and especially the Preservation Broker Contract, the transition of objects between systems, and the interaction between those systems, has reached a state where it is possible (and necessary) to establish agreements between the Active System, the PoF middleware, and the Digital Preservation System (DPS). These agreements are formulated in the Preservation Broker Contract and include parts that are relevant mainly for documentation, but the major parts handle information on what is possible to automate within the PoF framework, but also to some extent in the DPS and in the communication between systems. This means that the Active System user/owner has to spend some time initially on setting up this contract, but many of the parts are intended for the PoF manager/operator and thereby not something that should concern the customer.

6.1 Lessons Learned

The work in this workpackage has led us to start working on formalised, machine-readable, contracts handling, among other things, evolution of the environmental setting of these objects, including many-to-many relationships between systems. One lesson learned is that it would have gone smoother and gained more impact if the idea had been in the project from the beginning, but although that was not the case – we have still gained momentum in the development of the contract. The considerations that have to go into the process dealing with individuals (personal preservation) also gave us new insight into e.g. different needs for ownership, condensation or even removal of preserved objects that differ from typical organisational settings which usually are more formalised and role-based. The personal preservation setting could serve as an interesting input even to the more well defined archival setting that many, but not all, organisations have. The purpose, in the long run, is that the contract should be able to combine certain elements from both "fields" that in turn lead to a preservation solution that is somewhat tailor-made for each customer.

The work with a middleware, message queue, and eventually an Enterprise Service Bus (ESB), has been rewarding and although not necessarily novel, the service bus together with the Preservation Broker Contract can be used to specify different solutions (paths) for different customers. The idea is that the contract then holds information independently from the ESB, and thereby lessens the dependency on a certain service bus, but there is room for more development on this part, which leads us into the next section.

6.2 Vision for the Future

We already today see an increasing demand for managing many-to-many relationships in digital preservation, for example in agencies in Sweden. There exists an initial interest from preservation service providers, including at the level of government agencies, to continue working on the Preservation Broker Contract, adapting it to a national context if needed. This could for example include developing it further to support Business Process Modelling, e.g. through use of XML Process Definition Language (XPDL)[19].

[19] http://www.xpdl.org/

References

[Afrasiabi Rad, Nilsson, Päivärinta, 2014] Afrasiabi Rad, P., Nilsson, J., Päivärinta, T. (2014). Administration of Digital Preservation Services in the Cloud Over Time : Design Issues and Challenges for Organizations. *The Proceedings of the 2nd International Conference on Cloud Security Management*, The Proceedings of the 2nd International Conference on Cloud Security Management / edited by Barbara Endicott-Popovsky.

[CCSDS, 2004] Consultative Committee for Space Data Systems (2004). *Producer-Archive Interface Methodology Abstract Standard. Issue 1*. Recommendation for Space Data System Practices (Magenta Book), CCSDS 651.0-M-1. Washington, D.C.: CCSDS Secretariat, May 2004. [Equivalent to ISO 20652:2006.]

[CCSDS, 2014] Consultative Committee for Space Data Systems (2014). *Producer-Archive Interface Specification (PAIS)*. Recommendation for Space Data System Standards (Blue Book), CCSDS 651.1-B-1. Washington D.C.: CCSDS Secretariat, February 2014.

[CCSDS, 2012] Consultative Committee for Space Data Systems (2012). *Reference Model for an Open Archival Information System (OAIS). Issue 2.* Recommendation for Space Data System Practices (Magenta Book), CCSDS 650.0-M-2. Washington, D.C.: CCSDS Secretariat, June 2012. [Equivalent to ISO 14721:2012.]

[Chen et al., 2016] Chen, D., Gallo, F., Gür, G., Greenwood, M.A., Andersson, I., Nilsson, J. (2015). D7.4: Computational Storage Services – Third Release. ForgetIT.

[Gallo et al., 2014] Gallo, F., Pellegrino, J., Niederée, C., Kanhabua, N., Chen, D., Maus, H., Solachidis, V., Damhuis, A., Greenwood, M.A. (2014). *D8.3: The Preserve-or-Forget Framework – First release*. ForgetIT

[Gallo et al., 2015a] Gallo, F., Niederée, C., Andersson, I., Nilsson, J., Chen, D., Maus, H., Greenwood, M., and Logie, R. (2015). *D8.2: The Preserve-or-Forget Reference Model Initial Model*. ForgetIT

[Gallo et al., 2015b] Gallo, F., Ceroni, A., Tran, T., Chen, D., Andersson, I., Greenwood, M. A., Maus, H., ... Goslar, J. (2015). *Deliverable D8.4: The Preserve-or-Forget Framework - Second Release*. ForgetIT

[Gallo et al., 2016] Gallo, F., Niederée, C., Andersson, I., Nilsson, J., Chen, D., Maus, H., Greenwood, M. A., Logie, R., and Allasia, W. (2016). *Deliverable D8.5: The Preserve-or-Forget Reference Model - Final Model*. ForgetIT

[Greenwood et al., 2016] Greenwood, M. A., Petrak, J., Gorrell, G., Solachidis, V., Papadopoulou, O., Apostolidis, … Maus, H. (2016). *D6.4 Contextualisation Framework and Evaluation.* ForgetIT

[Kanhabua et al., 2015] Kanhabua, N., Niederée, C., Ceroni, A., Djafari-Naini, K., Kawase, R., Tran, T., Maus, H., Schwarz, S. (2015). *D3.3: Strategies and Components for Managed Forgetting – Second Release*. ForgetIT

[Linthicum, 2000] Linthicum, D. S. (2000). *Enterprise Application Integration. Upper Saddle River*, NJ: Addison-Wesley Professional.

[Mezaris et al., 2016] Mezaris, V., Solachidis, V., Chen, D., Eldesouky, B., Greenwood, M.A., Tan, A.S., … Tastzoglou., D. (2016). *D4.4 Information analysis, consolidation and concen-tration techniques, and evaluation - Final release*. ForgetIT

[Nilsson et al., 2014] Nilsson, J. Andersson, I, Afrasiabi Rad, P., Lindqvist, G., Gallo, F., Rabinovici-Cohen, S., Maus. H., Dobberkau, O., Allasia, W., Päivärinta, T. (2014). *D5.2*: *Workflow model and prototype for transition between active system and AIS - first release*. ForgetIT

[Nilsson et al., 2015] Nilsson, J., Andersson, I., Lindqvist, G., Westerlund, P. (2015). *D5.3: Workflow model and prototype for transition between active system and AIS – second release.* ForgetIT

[Papazoglou, 2003] Papazoglou, M. P. (2003). Service-oriented computing: Concepts, characteristics and directions. In *Web Information Systems Engineering, 2003. WISE 2003. Proceedings of the Fourth International Conference on* (pp. 3-12). IEEE.

[Preservica, 2014] Preservica (2014), *Tessella launches Preservica subsidiary business*. Press Release 2014-04-29, http://preservica.com/press-releases/tessella-launches-preservica-subsidiary-business/ Accessed 2015-12-12.

[Päivärinta et al., 2014] Päivärinta, T., Nilsson, J., Afrasiabi Rad, P., Maus, H., Dobberkau, O. (2014). *D5.1: Concise preservation by combining managed forgetting and contextualization remembering: Foundations of synergetic preservation*. ForgetIT.

[Wilkes et al. 2009] Wilkes, W., Brunsmann, J., Heutelbeck, D., Hundsdörfer, A., Hemmje, M., & Heidbrink, H. U. (2009). Towards support for long-term digital preservation in product life cycle management. *California Digital Library*.

[York, 2010] York, J. (2010). Building a future by preserving our past: the preservation infrastructure of HathiTrust digital library. In *76th IFLA general congress and assembly* (pp. 10-15).

Appendix A – Preservation Broker Contract XML Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
*************************************************************************************-->
<!-- First draft of PBC Schema, 2016-01-11, Göran Lindqvist, Ingemar Andersson, Jörgen
Nilsson Luleå University of Technology -->
<!-- pbc version 0.3 -->
<!--
*************************************************************************************-->

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
    attributeFormDefault="unqualified">
    <xs:element name="pbc" type="pbc"/>
    <xs:complexType name="pbc">
        <xs:sequence>
            <xs:element name="identification" type="identification"/>
            <xs:element name="collectionType" type="collectionType" minOccurs="0"/>
            <xs:element name="services" type="services" minOccurs="0"/>
            <xs:element name="actions" type="actions" minOccurs="0"/>
            <xs:element name="rules" type="rules"/>
        </xs:sequence>
    </xs:complexType>
    <!--*****Identification main nodes*****-->
    <xs:complexType name="identification">
        <xs:sequence>
            <xs:element name="pbcTemplateId" type="xs:string"/>
            <xs:element name="preservationLevel" type="preservationLevelValue"/>
            <xs:element name="pbcId" type="xs:string"/>
            <xs:element name="pbcCreationDate" type="xs:dateTime" default="2015-08-
01T00:00:00Z"/>
            <xs:element name="pbcValidFrom" type="xs:dateTime" default="2015-11-
01T00:00:00Z"/>
            <xs:element name="pbcExpirationDate" type="xs:dateTime" default="2020-12-
31T00:00:00Z"/>
            <xs:element name="pbcDescription" type="xs:string"/>
            <xs:element name="contact" type="contact"/>
            <xs:element name="activeSystem" type="activeSystem"/>
            <xs:element name="preservationSystem" type="preservationSystem"/>
            <xs:element name="preservationValue" type="xs:string"/>
        </xs:sequence>
    </xs:complexType>
    <!--Values: identification:preservationLevel-->
    <xs:simpleType name="preservationLevelValue">
        <xs:restriction base="xs:string">
            <xs:enumeration value="premium"/>
            <xs:enumeration value="standard"/>
            <xs:enumeration value="basic"/>
        </xs:restriction>
    </xs:simpleType>
    <!-- contact => Identification-->
    <xs:complexType name="contact">
        <xs:sequence>
            <xs:element name="ownerName" type="xs:string"/>
            <xs:element name="ownerEmail" type="xs:string"/>
            <xs:element name="alternativeContactName" type="xs:string"/>
            <xs:element name="alternativeContactEmail" type="xs:string"/>
        </xs:sequence>
    </xs:complexType>
    <!-- activeSystem => Identification-->
    <xs:complexType name="activeSystem">
        <xs:sequence>
```

```
            <xs:element name="Id" type="xs:string"/>
            <xs:element name="name" type="xs:string"/>
            <xs:element name="type" type="xs:string"/>
            <xs:element name="version" type="xs:string"/>
            <xs:element name="serviceId" type="xs:string"/>
            <xs:element name="ownedBy" type="xs:string"/>
        </xs:sequence>
    </xs:complexType>
    <!--preservationSystem => Identification-->
    <xs:complexType name="preservationSystem">
        <xs:sequence>
            <xs:element name="Id" type="xs:string"/>
            <xs:element name="name" type="xs:string"/>
            <xs:element name="version" type="xs:string"/>
            <xs:element name="location" type="xs:string"/>
            <xs:element name="serviceId" type="xs:string"/>
            <xs:element name="ownedBy" type="xs:string"/>
        </xs:sequence>
    </xs:complexType>
    <!--*****collectionType main nodes*****-->
    <xs:complexType name="collectionType">
        <xs:sequence>
            <!--<xs:element name="groupType" type="groupType" minOccurs="0"/> -->
            <xs:element name="label" type="xs:string" minOccurs="0"/>
            <xs:element name="descripton" type="xs:string" minOccurs="0"/>
            <xs:element name="quantity" type="quantity" minOccurs="0"/>
            <xs:element name="size" type="size" minOccurs="0"/>
            <xs:element name="mimeTypes" type="mimeTypes" minOccurs="0"/>
            <xs:element name="tranformations" minOccurs="0">
              <xs:complexType>
                    <xs:sequence>
                          <xs:element name="transformation" type="transformation"
minOccurs="0"
                                maxOccurs="unbounded"/>
                    </xs:sequence>
              </xs:complexType>
            </xs:element>
            <xs:element name="keepSourceFileName" type="keepSourceFileNameValue"
default="yes"
              minOccurs="0"/>
        </xs:sequence>
    </xs:complexType>
    <!--groupType => Collection-->
    <xs:complexType name="groupType">
        <xs:sequence>
            <xs:element name="Id" type="xs:string" minOccurs="0"/>
            <xs:element name="description" type="xs:string" minOccurs="0"/>
            <xs:element name="structureName" type="xs:string" minOccurs="0"/>
            <xs:element name="occurrence" type="occurrence" minOccurs="0"/>
        </xs:sequence>
    </xs:complexType>
    <!--groupTypeOccurrence => groupType -> Collection-->
    <xs:complexType name="occurrence">
        <xs:sequence>
            <xs:element name="min" type="xs:string" minOccurs="0"/>
            <xs:element name="max" type="xs:string" minOccurs="0"/>
        </xs:sequence>
    </xs:complexType>
    <!--quantity => Collection-->
    <xs:complexType name="quantity">
        <xs:sequence>
            <xs:element name="min" type="xs:string" minOccurs="0"/>
            <xs:element name="max" type="xs:string" minOccurs="0"/>
        </xs:sequence>
    </xs:complexType>
    <!--size => Collection-->
    <xs:complexType name="size">
        <xs:sequence>
            <xs:element name="min" type="xs:positiveInteger" minOccurs="0"/>
            <xs:element name="max" type="xs:positiveInteger" minOccurs="0"/>
            <xs:element name="unitType" type="xs:string" default="MB" minOccurs="0"/>
        </xs:sequence>
    </xs:complexType>
    <!--Values: Collection:size-->
    <xs:simpleType name="unitTypeValue">
        <xs:restriction base="xs:string">
            <xs:enumeration value="MB"/>
            <xs:enumeration value="GB"/>
        </xs:restriction>
    </xs:simpleType>
    <!--mimeTypes => Collection-->
    <xs:complexType name="mimeTypes">
        <xs:sequence>
            <xs:element name="mimeType" maxOccurs="unbounded" type="xs:string">
</xs:element>
        </xs:sequence>
    </xs:complexType>
    <!--formatTransformations => CollectionType-->
    <xs:complexType name="formatTransformations"/>
    <!--tranformation => formatTransformations => CollectionType-->
    <xs:complexType name="transformation">
        <xs:sequence>
            <xs:element name="onAction" type="onActionValue" minOccurs="0"/>
            <xs:element name="keepOriginal" type="keepOriginalValue" default="yes"
minOccurs="0"/>
            <xs:element name="origMimeType" type="xs:string" minOccurs="0"/>
            <xs:element name="origPuid" type="xs:string" minOccurs="0"/>
            <xs:element name="targetMimeType" type="xs:string" minOccurs="0"/>
            <xs:element name="targetPuid" type="xs:string" minOccurs="0"/>
            <xs:element name="serviceId" type="xs:string" minOccurs="0"/>
        </xs:sequence>
    </xs:complexType>
    <!--Values: Collection:CollectionType:formatTransformations:tranformation:onAccess-->
    <xs:simpleType name="onActionValue">
        <xs:restriction base="xs:string">
            <xs:enumeration value="ingest"/>
            <xs:enumeration value="access"/>
        </xs:restriction>
    </xs:simpleType>
    <!--*****services main nodes*****-->
    <xs:complexType name="services">
        <xs:sequence>
            <xs:element name="service" type="service" minOccurs="0"
maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>
    <!--service => services-->
    <xs:complexType name="service">
        <xs:sequence>
            <xs:element name="id" type="xs:string" minOccurs="0"/>
            <xs:element name="name" type="xs:string" minOccurs="0"/>
            <xs:element name="description" type="xs:string" minOccurs="0"/>
            <xs:element name="endpoint" type="xs:string" minOccurs="0"/>
            <xs:element name="user" type="xs:string" minOccurs="0"/>
            <xs:element name="password" type="xs:string" minOccurs="0"/>
            <xs:element name="endpointType" type="xs:string" minOccurs="0"/>
            <xs:element name="responseDataFormat" type="xs:string" minOccurs="0"/>
            <xs:element name="version" type="xs:string" minOccurs="0"/>
        </xs:sequence>
    </xs:complexType>
    <!--*****actions main nodes*****-->
    <xs:complexType name="actions">
        <xs:sequence>
            <xs:element name="onPreIngest" type="onPreIngest" minOccurs="0"/>
            <xs:element name="onAccess" type="onAccess" minOccurs="0"/>
            <xs:element name="onStorage" type="onStorage" minOccurs="0"/>
        </xs:sequence>
    </xs:complexType>
    <!--onPreIngest => actions-->
    <xs:complexType name="onPreIngest">
        <xs:sequence>
            <xs:element name="appraisalThresholds" type="appraisalThresholds"
minOccurs="0"/>
            <xs:element name="downloadStructure" type="downloadStructureValue"
default="none"
             minOccurs="0"/>
            <xs:element name="packageEncodingType" type="packagEncodingValue"
default="zip"
             minOccurs="0"/>
            <xs:element name="fixity" type="fixity" minOccurs="0"/>
            <xs:element name="externalMetadata" type="externalMetadata" minOccurs="0"/>
            <xs:element name="sip" type="sip"/>
        </xs:sequence>
    </xs:complexType>
    <!--Values: onPreIngest:packageEncodingType-->
    <xs:simpleType name="packagEncodingValue">
        <xs:restriction base="xs:string">
            <xs:enumeration value="tar"/>
            <xs:enumeration value="zip"/>
        </xs:restriction>
    </xs:simpleType>
    <!--Values: "onPreIngest:downloadStructure"-->
    <xs:simpleType name="downloadStructureValue">
        <xs:restriction base="xs:string">
            <xs:enumeration value="physical"/>
            <xs:enumeration value="logical"/>
            <xs:enumeration value="none"/>
        </xs:restriction>
    </xs:simpleType>
    <!-- appraisalThresholds => onPreIngest-->
    <xs:complexType name="appraisalThresholds">
        <xs:sequence>
            <xs:element name="preservationValue" type="xs:string" minOccurs="0"/>
        </xs:sequence>
    </xs:complexType>
    <!-- fixity => onPreIngest-->
    <xs:complexType name="fixity">
        <xs:sequence>
            <xs:element name="onRetrieval" type="onRetrieval" minOccurs="0"/>
            <xs:element name="atRest" type="atRest" minOccurs="0"/>
            <xs:element name="inPackage" type="inPackage"/>
            <xs:element name="onTransfer" type="onTransfer" minOccurs="0"/>
        </xs:sequence>
    </xs:complexType>
    <!--OnRetrieval => fixity -> onPreIngest-->
    <xs:complexType name="onRetrieval">
        <xs:sequence>
            <xs:element name="algorithm" type="algorithmValues" default="md5"
minOccurs="0"/>
```

```
            <xs:element name="serviceId" type="xs:string" minOccurs="0"/>
        </xs:sequence>
    </xs:complexType>
    <!--AtRest => fixity -> onPreIngest-->
    <xs:complexType name="atRest">
        <xs:sequence>
            <xs:element name="algorithm" type="algorithmValues" default="md5"
minOccurs="0"/>
            <xs:element name="serviceId" type="xs:string" minOccurs="0"/>
        </xs:sequence>
    </xs:complexType>
    <!--InPackage => fixity -> onPreIngest-->
    <xs:complexType name="inPackage">
        <xs:sequence>
            <xs:element name="algorithm" type="algorithmValues" default="md5"
minOccurs="0"/>
            <xs:element name="serviceId" type="xs:string" minOccurs="0"/>
        </xs:sequence>
    </xs:complexType>
    <!--OnTransfer => fixity -> onPreIngest-->
    <xs:complexType name="onTransfer">
        <xs:sequence>
            <xs:element name="algorithm" type="algorithmValues" default="md5"
minOccurs="0"/>
            <xs:element name="serviceId" type="xs:string" minOccurs="0"/>
        </xs:sequence>
    </xs:complexType>
    <!--Values: fixity:algorithm-->
    <xs:simpleType name="algorithmValues">
        <xs:restriction base="xs:string">
            <xs:enumeration value="md5"/>
            <xs:enumeration value="sha1"/>
            <xs:enumeration value="sha256"/>
        </xs:restriction>
    </xs:simpleType>
    <!-- externalMetadata => onPreIngest-->
    <xs:complexType name="externalMetadata">
        <xs:sequence>
            <xs:element name="sources" type="sources" minOccurs="0"/>
        </xs:sequence>
    </xs:complexType>
    <!-- sources => externalMetadata -> onPreIngest-->
    <xs:complexType name="sources">
        <xs:group ref="sourceFilters" minOccurs="0" maxOccurs="unbounded"/>
    </xs:complexType>
    <!--group source and filters-->
    <xs:group name="sourceFilters">
        <xs:sequence>
            <xs:element name="serviceId" type="xs:string" minOccurs="0"/>
            <xs:element name="parameters" type="parameters" minOccurs="0"/>
        </xs:sequence>
    </xs:group>
    <!--parameters =>sourceFilter(group) -> sources -> externalMetadata -> onPreIngest-->
    <xs:complexType name="parameters">
        <xs:sequence>
            <xs:element name="parameter" type="xs:string" minOccurs="0"/>
        </xs:sequence>
    </xs:complexType>
    <!--onIngest => actions-->
    <xs:complexType name="onIngest">
        <xs:sequence>
            <xs:element name="formatTransformation" type="formatTransformations"
minOccurs="0"/>
```

```
        </xs:sequence>
    </xs:complexType>
    <!--Values: onIngest:keepSourceFileName-->
    <xs:simpleType name="keepSourceFileNameValue">
        <xs:restriction base="xs:string">
            <xs:enumeration value="yes"/>
            <xs:enumeration value="no"/>
        </xs:restriction>
    </xs:simpleType>
    <!--Values: transformation:keepOrigin-->
    <xs:simpleType name="keepOriginalValue">
        <xs:restriction base="xs:string">
            <xs:enumeration value="yes"/>
            <xs:enumeration value="no"/>
        </xs:restriction>
    </xs:simpleType>
    <!--onAccess => actions-->
    <xs:complexType name="onAccess">
        <xs:sequence>
            <xs:element name="accessControl" type="accessControlValues" default="none"
minOccurs="0"/>
            <xs:element name="adaptStructure" type="adaptStructureValues" default="none"
              minOccurs="0"/>
            <xs:element name="externalMetadata" type="externalMetadata" minOccurs="0"/>
            <xs:element name="fixity" type="fixityOnAccess" minOccurs="0"/>
        </xs:sequence>
    </xs:complexType>
    <!--Values: onAccess:accessControl-->
    <xs:simpleType name="accessControlValues">
        <xs:restriction base="xs:string">
            <xs:enumeration value="none"/>
            <xs:enumeration value="medium"/>
            <xs:enumeration value="high"/>
        </xs:restriction>
    </xs:simpleType>
    <!--Values: onAccess:adaptStructure-->
    <xs:simpleType name="adaptStructureValues">
        <xs:restriction base="xs:string">
            <xs:enumeration value="none"/>
            <xs:enumeration value="logical"/>
            <xs:enumeration value="physical"/>
        </xs:restriction>
    </xs:simpleType>
    <xs:complexType name="fixityOnAccess">
        <xs:sequence>
            <xs:element name="algorithm" type="algorithmValues" minOccurs="0"/>
            <xs:element name="serviceId" type="xs:string" minOccurs="0"/>
        </xs:sequence>
    </xs:complexType>
    <!--onStorage => actions-->
    <xs:complexType name="onStorage">
        <xs:sequence>
            <xs:element name="level" type="levelValues" default="silver" minOccurs="0"/>
            <xs:element name="location" type="locationValues" default="none"
minOccurs="0"/>
            <xs:element name="size" type="storageSize" minOccurs="0"/>
            <xs:element name="versioning" type="xs:positiveInteger" minOccurs="0"/>
            <xs:element name="retentionPeriod" type="retentionPeriod" minOccurs="0"/>
            <xs:element name="fixity" type="checksum" minOccurs="0"/>
            <xs:element name="multiFactorDelete" type="keepOriginalValue" minOccurs="0"/>
            <xs:element name="encryption" type="encryption" minOccurs="0"/>
            <xs:element name="eventLogs" type="eventLogs" minOccurs="0"/>
            <xs:element name="eventNotification" type="eventNotification" minOccurs="0"
maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>
    <!--Values: onStorage:level-->
    <xs:simpleType name="levelValues">
        <xs:restriction base="xs:string">
            <xs:enumeration value="platinum"/>
            <xs:enumeration value="gold"/>
            <xs:enumeration value="silver"/>
            <xs:enumeration value="bronze"/>
        </xs:restriction>
    </xs:simpleType>
    <!--Values: onStorage:location-->
    <xs:simpleType name="locationValues">
        <xs:restriction base="xs:string">
            <xs:enumeration value="local"/>
            <xs:enumeration value="EU"/>
            <xs:enumeration value="US"/>
            <xs:enumeration value="none"/>
        </xs:restriction>
    </xs:simpleType>
    <!--storageSize => onStorage -> actions-->
    <xs:complexType name="storageSize">
        <xs:sequence>
            <xs:element name="max" type="xs:integer" minOccurs="0"/>
            <xs:element name="unitType" type="unitTypeValue" default="GB" minOccurs="0"/>
        </xs:sequence>
    </xs:complexType>
    <!--retentionPeriod => onStorage -> actions-->
    <xs:complexType name="retentionPeriod">
        <xs:sequence>
            <xs:element name="min" type="xs:integer" minOccurs="0"/>
            <xs:element name="nextLevel" type="levelValues" minOccurs="0"/>
            <xs:element name="nextLocation" type="locationValues" minOccurs="0"/>
            <xs:element name="max" type="xs:integer" minOccurs="0"/>
            <xs:element name="unitType" type="unitTypeValue" minOccurs="0"/>
            <xs:element name="endOfRetentionAction" type="xs:integer" minOccurs="0"/>
        </xs:sequence>
    </xs:complexType>
    <!--fixity => onStorage -> actions-->
    <xs:complexType name="checksum">
        <xs:sequence>
            <xs:element name="algorithm" type="algorithmValues" minOccurs="0"/>
            <xs:element name="Interval" type="xs:integer" minOccurs="0"/>
            <xs:element name="intervalUnit" type="xs:string" minOccurs="0"/>
        </xs:sequence>
    </xs:complexType>
    <!--encryption => onStorage -> actions-->
    <xs:complexType name="encryption">
        <xs:sequence>
            <xs:element name="keyAtRest" type="xs:string" minOccurs="0"/>
            <xs:element name="keyAtTransfer" type="xs:string" minOccurs="0"/>
        </xs:sequence>
    </xs:complexType>
    <!--eventLogs => onStorage -> actions-->
    <xs:complexType name="eventLogs">
        <xs:sequence>
            <xs:element name="event" type="xs:string" minOccurs="0"
maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>
    <!--eventNotifications => onStorage -> actions-->
    <xs:complexType name="eventNotification">
        <xs:sequence>
            <xs:element name="eventType" type="xs:string" minOccurs="0"/>
            <xs:element name="serviceId" type="xs:string" minOccurs="0"/>
        </xs:sequence>
    </xs:complexType>
    <!--*****ruleSets main nodes*****-->
    <!--ruleSet -> ruleSets-->
    <!--rule -> ruleSet-->
    <xs:complexType name="rule">
        <!--            <xs:attribute name="name" type="xs:string"/>
            <xs:attribute name="description" type="xs:string"/> -->
        <xs:sequence>
            <xs:element name="Id" type="xs:string"/>
            <xs:element name="type" type="xs:string"/>
            <xs:element name="description" type="xs:string"/>
            <xs:element name="ruleparameters" type="ruleparameters"/>
            <xs:element name="action" type="xs:string"/>
            <xs:element name="otherwiseAction" type="xs:string"/>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="ruleparameters">
        <xs:sequence>
            <xs:element name="ruleparameter" type="ruleparameter" minOccurs="0"
              maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="ruleparameter">
        <xs:sequence>
            <xs:element name="name" type="xs:string"/>
            <xs:element name="dependency" type="xs:string"/>
            <xs:element form="qualified" name="condition" type="xs:string"/>
            <xs:element name="value" type="xs:string"/>
        </xs:sequence>
    </xs:complexType>
    <xs:element name="file" type="xs:string"/>
    <xs:element name="folder" type="xs:string"/>
    <xs:complexType name="packageStructure">
        <xs:sequence>
            <xs:element maxOccurs="unbounded" ref="file"/>
            <xs:element maxOccurs="unbounded" ref="folder"/>
        </xs:sequence>
    </xs:complexType>
    <xs:element name="metadataSchema" type="xs:string"/>
    <xs:complexType name="sip">
        <xs:sequence>
            <xs:element maxOccurs="unbounded" ref="metadataSchema"/>
            <xs:element name="packageStructure" type="packageStructure"/>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="rules">
        <xs:sequence>
            <xs:element maxOccurs="unbounded" name="rule" type="rule" form="qualified"/>
        </xs:sequence>
    </xs:complexType>
</xs:schema>
```