www.forgetit-project.eu

ForgetIT

Concise Preservation by Combining Managed Forgetting and Contextualized Remembering

Grant Agreement No. 600826

Deliverable D8.1

Work-package	WP8: PoF Reference Model and Framework
Deliverable	D8.1: Integration Plan and Architectural Approach
Deliverable Leader	Francesco Gallo (EURIX)
Quality Assessor	Heiko Maus (DFKI)
Estimation of PM spent	9
Dissemination level	PU
Delivery date in Annex I	31. October 2013 (M9)
Actual delivery date	6. December 2013
Revisions	7
Status	Final
Keywords:	Preserve-or-Forget Architecture, Reference Model, Framework, Integration

Disclaimer

List of Authors

Partner Acronym	Authors
LUH	Claudia Niederée
LUH	Kaweh Djafari-Naini
LTU	Ingemar Andersson
LTU	Parvaneh Afrasiabi Rad
LTU	Göran Lindqvist
LTU	Jörgen Nilsson
IBM	Simona Rabinovici-Cohen
IBM	Ealan Henis
DFKI	Heiko Maus
DFKI	Frank Steinmann
CERTH	Vasileios Mezaris
CERTH	Olga Papadopoulou
CERTH	Vasilis Solachidis
dkd	Olivier Dobberkau
dkd	Phuong Doan
EURIX	Walter Allasia
EURIX	Francesco Gallo

Contents

Executive summary

The primary objectives of WP8 are (a) to devise a reference model which comprises the concepts and processes for managed forgetting, contextualized remembering and for integrated information and preservation management (synergetic preservation) and (b) based on this reference model to integrate the components, which implement these concepts, into a technologically coherent framework, the Preserve-or-Forget (PoFPoF) framework.

This document describes the architecture of the PoF framework with its main components. The components developed by the technical WPs (WP3-WP7) are integrated into the framework. In this deliverable we also describe the integration plan for testing and validating the project results. The work presented in this document are the results of Task 8.1.

Making use of Model Driven Architecture (MDA) approach (and UML notation), under the lead of EURIX, all technical partners have worked together to establish and document a first version of the functional specifications of the ForgetIT architecture and the corresponding integration guidelines for the different components.

The architecture defined in the present document will be improved in an iterative approach, to be refined and extended during the project when new results and insight will become available for integration and testing. In this deliverable we also included a preliminary evaluation of platforms and other candidate solutions for the implementation of the building blocks in the architecture.

The defined architecture includes two active systems (the Semantic Desktop and TYPO3, see WP9 and WP10, respectively), an Archive compliant to OAIS model, a PoF Middleware providing the bridge between the active systems and the Archive and finally a Cloud Storage Service with a Storlet engine for executing specific tasks close to the data. The Archive and the Middleware will exploit existing solutions wherever possible, which will be adapted and customized within WP8. Furthermore, the PoF Middleware will integrate components developed by WP3-WP6, which implement the core ForgetIT functionality. The cloud storage system will be developed in WP7.

1 Introduction

The first objective of WP 8 is to define - in collaboration with the other technical WPs as well as with the interdisciplinary components from WP2 - a reference model which comprises the concepts and processes for managed forgetting, contextualized remembering and synergetic preservation. As support for this reference model, the second objective is to integrate the components developed in the project into a technologically coherent framework which can be used to implement the Preserve-or-Forget (PoF) Reference Model. The overall approach will be based on flexible and extensible solutions making use of the most appropriate technologies concerning protocols, metadata and formats.

This document describes the architecture of the Preserve-or-Forget (PoF) framework with its main components providing a high level description of each module, showing relevant interfaces and protocols used for communication among them. The components developed by the technical WPs will be integrated into this framework. In this deliverable we also describe the integration plan for testing and validating the project results.

The present document is the result of the activity performed in Task 8.1 - Integration plan and ForgetIT architecture, whose main objective is to ensure that (1) the components developed in WP3-WP7 work together and that (2) the conceptual architecture for the PoF framework will be specified, including component responsibilities, interface definitions and the foreseen interplay between components. Task 8.1 also targets the plan for the integration of software components into the ForgetIT architecture.

The defined architecture includes two active systems (the Semantic Desktop and TYPO3), an Archive compliant to OAIS model, a PoF Middleware implementing the ForgetIT intelligent preservation solution and providing the bridge between the active systems and the Archive and finally a Cloud Storage Service with a Storlet engine for executing specific tasks close to the data.

Wherever possible, the Archive and the Middleware will rely on existing solutions, which will be adapted and customized within WP8. The PoF Middleware will integrate components developed by WP3-WP6 for realizing the novel ForgetIT methods. The cloud storage system will be developed in WP7.

The architecture defined in the present document will be improved in an iterative approach during the project.

Previous deliverables provide an input to the present document. In particular, D3.1 [1] discusses the importance of managed forgetting and provides useful guidelines for the PoF platform. Other deliverables, such as D4.1 [2], D5.1 [3], D6.1 [4] and D7.1 [5], describe the foundations and the state of the art for relevant topics such as information extraction, synergetic preservation, contextualization and computational storage services.

Based on the results provided by these documents, several components will be developed during the project lifetime: this document provides information about how such component

will work together in an integrated environment.

Finally, D9.1 [6] introduces the two main application use cases and scenarios, which have been taken into account when designing the architecture. Two simplified scenarios based on D9.1 will be discussed: they will be used as first demonstration of early integration.

This document, on the other hand, will provide input to future deliverables, including not only WP8 deliverables related to the Reference Model or the different platform releases, but also for example D3.2, D4.2 and D6.2, which will present the first prototypes of the components.

Another relevant deliverable will be D5.2 [7], where a workflow model for transition between active systems and the archive will be discussed: this document will be based on the architecture described here and will provide an input to the future deliverables about the PoF platform.

The document is organized as follows: in Section 2 we provide an overview of the overall architecture, with the main components; in Section 3 we provide a detailed description of the components which will be developed in the project and integrated in the PoF Framework; in Section 4.1, making use of UML2 notation, the structural representation of the architecture is presented, as well as two priority workflows for the initial integration activities for the PoF framework; in Section 5 we describe different solutions compliant to OAIS which are possible candidates for the implementation of the Archive in our architecture; in Section 6 a preliminary overview of candidate technologies for the middleware is provided; in Section 7 we discuss the integration plan for the different releases of the platform; in Section 8 we describe the test environment which will be used to develop and test the integrated components and systems of the platform; finally we provide a Glossary with a list of acronyms and abbreviations used in the document, referring to special terms in the field of digital preservation and other technologies used and developed in the project.

2 ForgetIT Architecture

The architecture of the PoF framework is the first outcome of WP8. The main purpose was to design a system to integrate all components developed in the project into a technologically coherent framework which could be used to implement the PoF Reference Model. In order to achieve this, the expected results of each technical WP have been evaluated and a collaborative approach has been adopted to design the overall architecture. The method used to design the architecture is based on MDA and makes use of UML2 as the standard modelling language for designing the architecture since the preliminary sketches to the final representation, using an iterative approach.

The ForgetIT architecture is made up of four main layers, which are shortly described in the following Sections. The first layer includes the active systems, namely the Semantic Desktop (see WP9) and the TYPO3 CMS (see WP10). These two applications are related to the main scenarios, as already described in D9.1 [6]. The second layer is provided by what we call the *PoF Middleware* (Preserve-or-Forget) Middleware, whose main purpose is to enable seamless transition from active systems to the Archive (and vice versa) and to provide the necessary functionality for supporting managed forgetting and contextualized remembering. The middleware will integrate components developed in WP3, WP4, WP5, and WP6, which implement the ForgetIT functionalities. The third layer is made up of the Archive, which is an OAIS compliant platform responsible for the digital preservation of the contents created by the applications and implements the backbone for synergetic preservation. The Archive provides ingest and access functionalities, as well as data management for the archived content, data curation for actual preservation and content storage. The development and integration of the Archive is under the responsibility of WP8. The fourth layer corresponds to the cloud storage, which will be the main outcome of WP7. The approach adopted in ForgetIT to implement the Archival Storage functionality will make use of an advanced solution for cloud storage powered by a mechanism to execute resource consuming tasks close to the data.

The components are discussed in the next Sections. The information reported for all architecture blocks demonstrates that all technical WPs will contribute to the integrated platform and that the designed architecture can accommodate all components in a coherent way. The integration mechanism when moving from one layer to the other has been identified (the expected interfaces and protocols have been selected, in some cases they are also available, for other components the work is still in progress).

An overview of the PoF architecture is depicted in Figure 1, which provides a graphical representation of the main components with some descriptive information (components developed during the project are shown in green, components shown in blue or cyan belong to existing platforms which will be further developed and customized to fit with project purposes and for integration in the overall architecture). The UML diagrams of the architecture can be found in Section 4.1.

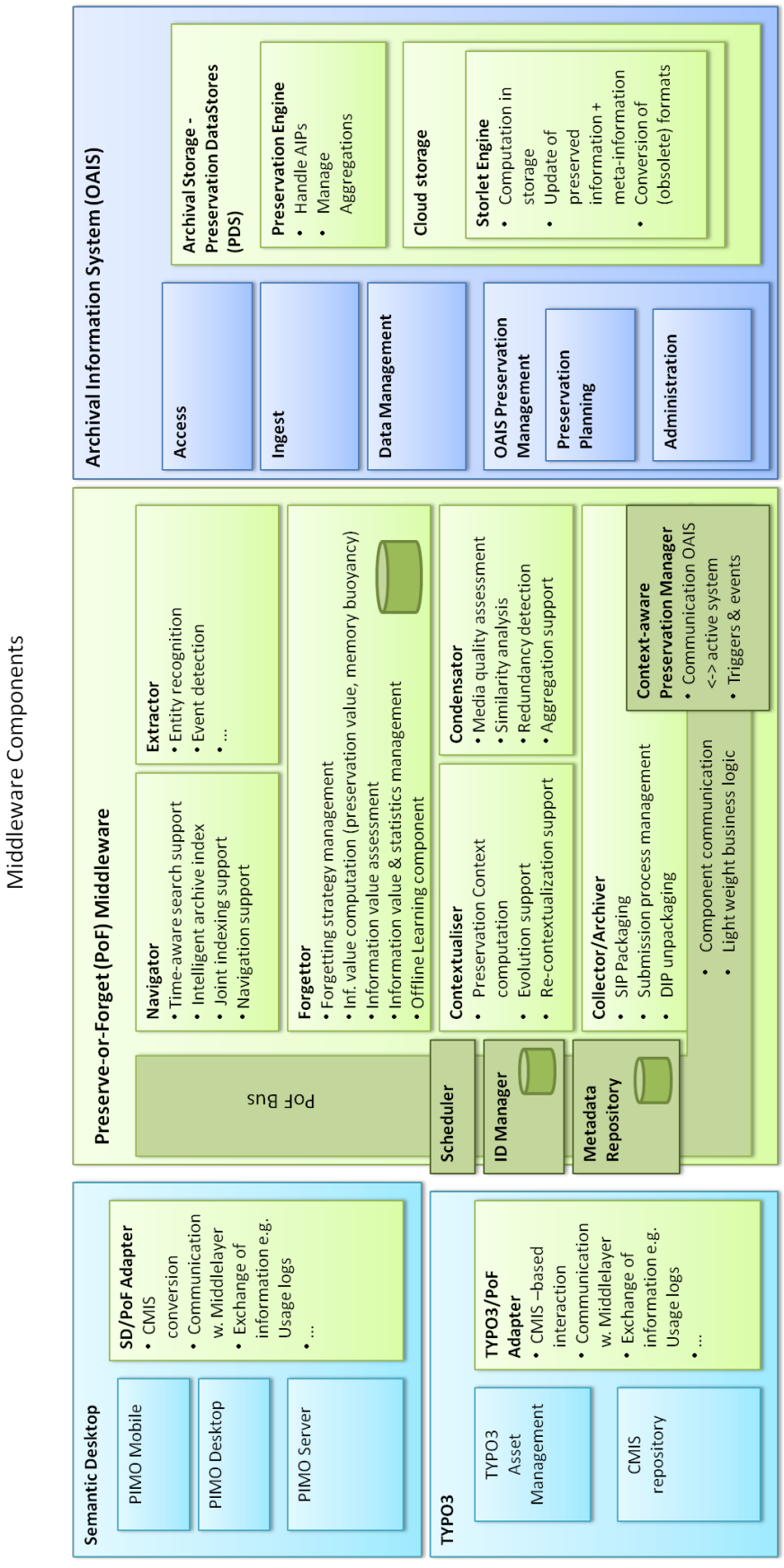

Figure 1: Overview of PoF architecture (*green*: components developed during the project; *blue*: existing components to be improved and customized.)

2.1 Active Systems

The PoF architecture includes two user applications which are used to validate the main scenarios. One is a personal preservation application based on the Semantic Desktop, developed in WP9 and the other one is an organizational preservation application based on the TYPO3 CMS, developed in WP10 (see D9.1 [6]). Both systems are complex and include several components, but the inner details are not discussed here (see deliverables D9.2 and D10.1 for details). Two example workflows involving the two applications as well as functionalities of other ForgetIT components are described in Section 4.2. Concerning the interfaces and the integration with the other architecture components, the two applications will make use of the REST APIs provided by the middleware to notify the system about content to be preserved and to retrieve updated content. Both components will expose a CMIS [8] interface to allow the middleware components to retrieve the content. The communication with the middleware is provided by application specific adapters, which encapsulate application-specific extensions for interacting with the PoF Middleware. The separation of functionalities that will be part of the Middelware and functionality, which will become part of the active systems is a challenging task. The details of this separation are still under discussion.

2.2 Middleware for synergetic preservation, managed forgetting and contextualized remembering

The main purpose of the middleware is to enable the three main principles of ForgetIT, namely a seamless transition from active systems to the archive (Synergetic Preservation), a meaningful transition back to the active system (Contextualized Remembering) as well as Managed Forgetting. The middleware includes a number of components for feature extraction, contextualization, condensation and managed forgetting. Other components are related to common tasks and are associated to the middleware bus. Examples are the scheduler and the ID manager. The integration of all the components into the middleware supporting the project scenarios will be the main challenge for the future integration activities in WP8. The middleware will expose REST APIs to be consumed by the applications and will include a CMIS client to retrieve content. Furthermore, the middleware includes components for importing and exporting content into and from the archive. The middleware will be able to properly package the content and associated metadata into a format which is compliant to what is expected by the archive.

2.3 Archive

The Archive implements the OAIS model [9], which is widely accepted as the reference standard for implementing digital preservation platforms. The Archive exposes different APIs for ingest and access, depending on the actual implementation. In addition to defining the parties involved in the long-term preservation of digital materials, OAIS provides

an information model for managing the digital materials as they pass through the system. A significant component of this model is the Information Package (IP). Each IP consists of the digital object(s) to be preserved, the metadata required at that point in the system and the Packaging Information which relates content and metadata (see [9]). OAIS outlines three types of Information Package: Submission Information Package (SIP), Archival Information Packages (AIP) and Dissemination Information Package (DIP). SIP and DIP are external to the archive and refer to the producer and consumer, respectively. AIP is internal to the archive. In ForgetIT, the active systems (via the PoF Middleware) act both as producers and as consumers. It is common practice to adopt the same representation for SIP, AIP, and DIP (e.g. several platforms use METS as XML wrapper for the three of them, but this is not mandatory). The OAIS functional model is depicted in Figure 2. For this architecture layer it is planned to rely on existing solutions. A list of candidate solutions is discussed in Section 5. The Archive must support the ForgetIT Reference Model and is responsible for the actual digital content curation. In the specific context of ForgetIT project, one of the OAIS functional entities, the Archival Storage, is implemented by a Cloud Storage Service.

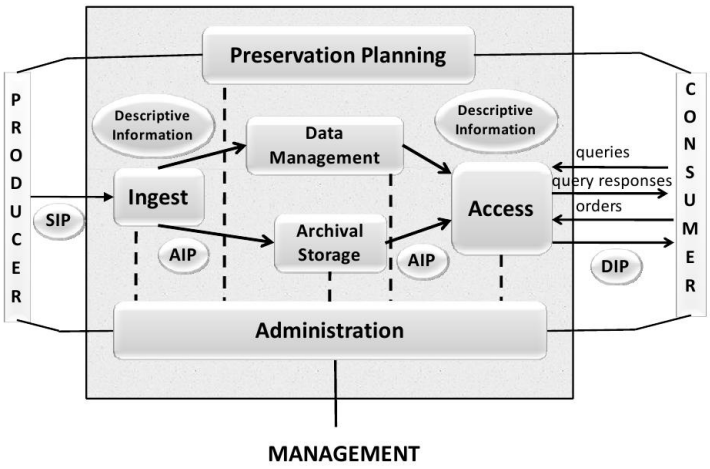

Figure 2: OAIS functional entities [9].

2.4 Cloud Storage Services

The cloud storage service integrated in the ForgetIT platform is based on Preservation DataStores (PDS) and OpenStack Swift, discussed in deliverable D7.1 [5], provides storage resources for AIPs but also a Storlet engine for executing specific operations close to the data, i.e. future processing steps can be done in the archive without requiring to extract it to a server and put it back into the archive. These tasks can include different content transformations, such as format migration, other resource consuming tasks, such as integrity checks as well as operations for enabling managed forgetting within the archive and operations for supporting context evolution for archived content.

3 Architecture Components

In this Section we describe the components which are part of the overall architecture, split into four main blocks (middleware, active systems, OAIS archive, cloud storage). The main components in the PoF platform are shown in Figure 1 and Figure 4. The middleware components are divided into (a) shared components (performing common tasks and managed through the middleware bus) and (b) components implementing the core ForgetIT functionalities, described in Section 3.2 and Section 3.3, respectively. For each component a fact sheet is provided, describing main functionalities and reporting information which is relevant for the integration.

3.1 Active Systems

The components related to the active systems are part of the current implementation of TYPO3 and Semantic Desktop together with the application-specific adapters. Only components relevant to the integration are considered in this deliverable. TYPO3 and the Semantic Desktop are described in more detail in deliverable D9.1 [6]. The main features of the two systems, in the context of their integration in the overall architecture are summarized in Table 1 and Table 2.

Component Name	**Semantic Desktop Infrastructure**
Partner Responsible	DFKI
Contributing Partners	
Work Packages	WP9 (WP3,WP4,WP5,WP8)
Reference Deliverables	D9.2, D9.3, D9.4
Current Status	Prototype available
Short description and role	The Semantic Desktop is a personal information management system with an underlying ontology semantically describing the user's mental model and the resources involved. This ontology is the PIMO (Personal Information Model). The Semantic Desktop infrastructure consist of a PIMO Server with a dedicated API so that any third party could use the PIMO for own purposes (e.g., using it as tagging vocabulary). In addition, a combination of plug-ins for (some) standard applications as well as dedicated components/UI for specific purposes (such as task management, photo collection) is provided. In ForgetIT the Semantic Desktop serves as a means to learn about user's resources, their usage over time, importance, interrelations, and context for each resource from the PIMO. Once the ForgetIT services are combined with the Semantic Desktop infrastructure, synergetic preservation is realized with nearly no additional effort. The PIMO will also provide context information for realizing contextual remembering as well as means for contributing to managed forgetting. The infrastructure will be enhanced with several ForgetIT services such as image quality assessment or text condensation.
Delivery Mode	Platform running in application server on dedicated machine
Subcomponents	PIMO Server, PIMO desktop clients, specific plug-ins for applications, HTML5 mobile client, User Observation Hub (UOH)
Main APIs, input and output formats	PIMO API (JSON RPC)
Plan for integration at M18	Platform connected to PoF Middleware; initial workflows are served
Plan for integration at M27/M36	Iterative enhancement of interplay with PoF Middleware; concise preserving desktop client M27; concise preserving mobile client M36
Language, runtime framework	server: Java, JSP, Apache Tomcat, MySQL; desktop: Java, HTML5 (JavaScript, CSS); mobile: HTML5
SW and HW Requirements	can be deployed on a VM
Dataset for testing	Stainer data set; 24/7 instance at DFKI (live usage); test instance for ForgetIT; cloning of PIMOs possible
License	a BSD-compliant license for interfaces and PIMO model; implementation free use in ForgetIT
Notes	The prototype and some documentation can be found at `https://pimo.kl.dfki.de/`

Table 1: Semantic Desktop (Active System, WP9 Component)

Component Name	**TYPO3**
Partner Responsible	dkd
Contributing Partners	
Work Package	WP10
Reference Deliverables	D5.1, D9.1, D10.1, D10.2, D10.3
Current Status	Design of Pilot Applications; Evaluation of Standards to be used
Short description and role	TYPO3 is an enterprise-class, Open Source CMS (Content Management System), used internationally to build and manage websites of all types, from small sites for non-profits to multilingual enterprise solutions for large corporations. TYPO3 is a user-friendly, intuitive tool for producing and maintaining web pages with just a few clicks of the mouse. Authors benefit from the full-featured rich-text editor that offers all of the formatting options they would need in a WYSIWYG (What You See Is What You Get) tool with a familiar word processor-like interface. Seamless integration of multimedia content and dynamic image manipulation are available right out of the box in TYPO3. In addition, an internal messaging and workflow system helps content creators and editors to collaborate in the administration backend. TYPO3 provides an extremely detailed permissions system for implementing professional editing workflows for both users and groups. Administrators can even manage multiple websites in one TYPO3 installation and share users, extensions, and content between them. Source code available on project repository [10]
Delivery Mode	Release in an agile approach
Subcomponents	CMIS, semantic annotations, ForgetIT TYPO3 Extensions (Content dashboard, Metadata directory, Semantic layer, Forget-IT module, Feedback and conflicts module, Recycle and inducing module, CMIS)
Main APIs, input and output formats	CMIS, REST, OWL
Plan for integration at M18	Application connected to PoF Middleware; initial workflows are served; See also evaluation plan
Plan for integration at M27/M36	Release of the pilot application with the ForgetIT TYPO3 Extensions and the integrated component ens (CMIS Client / Server and Semantic Services)
Language, runtime framework	TYPO3 CMS 6.2 LTS
SW and HW Requirements	Requirements for SW and HW are available in the project documentation (http://typo3.org/about/typo3-the-cms/system-requirements)
Dataset for testing	(1) Approved Spielwarenmesse press release (2) Testbed including multi domains in TYPO3 (3) Any other press releases and content consisting of text and media assets created in Testbed
License	GPL

Table 2: TYPO3 CMS (Active System, WP10 Component)

3.2 Shared Components of the PoF Middleware

The components listed below are integrated in the PoF Middleware. Shared components (managed by the Middleware bus) are responsible for general tasks.

Component Name	**Metadata Repository**
Partner Responsible	EURIX
Contributing Partners	L3S, USFD, CERTH, LTU
Work Packages	WP8, WP3, WP4, WP6, WP5
Reference Deliverables	D8.2
Current Status	Started component design
Short description and role	Component that manages metadata extracted or computed for individual documents and collections and makes them available for other components. This, for example, includes extracted entities, context information or memory buoyancy and preservation values. The metadata repository relies on the fact that all resource can be identified by an unique ID, which enable the retrieval of metadata stored in the repository for a resource. The repository might also include pointers to summaries for individual documents and/or document collections.
Delivery Mode	REST service
Subcomponents	Metadata storage, access/search support
Main APIs, input and output formats	Main methods include storage of new metadata for a resource, deletion of metadata for a resource, access to specific types of metadata given a resource and search in the metadata repository
Plan for integration at M18	Basic implementation according to the implemented scenarios, to support integration
Plan for integration at M27/M36	Improvements and extensions of repository
Language, runtime framework	Not decided yet, maybe re-use existing Open Source component and integrate it into the middleware
SW and HW Requirements	Requires database for managing the metadata
Dataset for testing	Some simple test cases should be sufficient
License	Open Source, if not otherwise implied by using an existing component

Table 3: Metadata Repository (Middleware, Shared Component)

Component Name	**ID Manager**
Partner Responsible	L3S
Contributing Partners	EURIX, dkd, DFKI
Work Packages	WP8, WP5 (interaction with WP9 and WP10 for identifier formats)
Reference Deliverables	D8.2, D5.2
Current Status	Started component design
Short description and role	This components mediates between the IDs used in the archive and the IDs used in the active systems. It might also be used to acquire new unique IDs.
Delivery Mode	REST service or API
Subcomponents	ID Repository, ID Generator
Main APIs, input and output formats	Main methods include generation of new ID and retrieval of IDs from a repository. Different standards can be used, such as UUID.
Plan for integration at M18	Basic implementation according to the implemented scenarios, to support integration
Plan for integration at M27/M36	Maybe improvements and extensions if necessary.
Language, runtime framework	Not decided yet, maybe re-use existing Open Source component to be integrated in the middleware
SW and HW Requirements	Requires database for managing the IDs
Dataset for testing	Some simple test cases should be sufficient
License	Open Source, if not otherwise implied by using an existing component

Table 4: ID Manager (Middleware, Shared Component)

Component Name	**Scheduler**
Partner Responsible	EURIX
Contributing Partners	L3S, LTU
Work Packages	WP8, WP3 (for scheduling of forgetting process), WP5 (for scheduling of archiving process)
Reference Deliverables	D8.2
Current Status	Started component design
Short description and role	Component that starts processes such as the forgetting process or an archiving process based on a defined schedule (e.g. once a day) or based on events, for which it is listening, plus additional conditions.
Delivery Mode	Active component, triggers other components and processes
Subcomponents	scheduling queue, event management
Main APIs, input and output formats	API that allows the scheduling of processes based on time and events, API for requesting status information, API for deletion of scheduled events
Plan for integration at M18	Basic implementation for simple time-based scheduling of processes and tasks
Plan for integration at M27/M36	Improvements and extensions
Language, runtime framework	Not decided yet, maybe re-use existing Open Source component to be integrated in the middleware
SW and HW Requirements	Not yet specified, depending on the actual implementation
Dataset for testing	Some scheduling test cases
License	Open Source, if not otherwise implied by using an existing component

Table 5: Scheduler (Middleware, Shared Component)

Component Name	**Context-Aware Preservation Manager**
Partner Responsible	LTU
Contributing Partners	LTU, EURIX, dkd, DFKI, TT, IBM
Work Packages	WP5
Reference Deliverables	D5.3
Current Status	Under development
Short description and role	The function of the Preservation Planning entity, and to some extent the Administration entity, in the AIS (Archive Information System / Preservation System) need to be ”stretched out” to meet the active systems (and their owners). Some of it is available through other components in the PoF middleware, but there still exists a need to handle changes on both sides of the middleware, which includes enabling communication of events and triggers relevant for both the preservation systems and the (owners of the) active systems. As an example, the preservation systems have internal preservation plans which might include transformation of classes of objects at ingest, if the objects are in unsuitable formats. These ”default transformations” need to be communicated to the active system. This may be communicated already at (or before) initial ingest of the first object of a specific type, since these plans are known beforehand. As another example, the AIS is responsible for preservation of the objects for long term, but the ForgetIT system must be able to re-contextualize the objects into active systems. This means that when the AIS make a decision to transform a class of objects - this must be communicated to the active system and its owners. If this transformation would ruin the chances of re-contextualization, e.g. by deleting the original object, some actions need to be taken to ensure the possibility of re-contextualization (e.g. transformation to another format for re-contextualization).
Delivery Mode	REST service
Subcomponents	Event logger
Main APIs, input and output formats	JSON, XML
Plan for integration at M18	Transmission of simple events between active system and AIS
Plan for integration at M27/M36	Communicate changes in AIS and active systems, regarding e.g. information structure
Language, runtime framework	Java/J2EE, Java App Server/Tomcat
SW and HW Requirements	Linux
Dataset for testing	Transformation event description in JSON or XML
License	Open Source

Table 6: Context-Aware Preservation Manager (Middleware, Shared Component)

3.3 Middleware components supporting core ForgetIT functionality

In addition to the shared components, which are part of the PoF Bus, the PoF Middleware contains seven components, which implement core ForgetIT functionality (see Figure 1):

- the **Forgettor** Component (see Table 7)
- the **Extractor** Component (see Table 8)
- the **Condensator** Component (see Table 9)
- the **Contextualizer** Component (see Table 10)
- the **Navigator** Component (see Table 11)
- the **Collector**/**Archiver** Component (see Table 12)

Together with the preservation system these components implement the three core ForgetIT principles of managed forgetting, contextualized remembering and synergetic preservation. Each of these components is described in a separate Table below.

Component Name	**Forgettor**
Partner Responsible	L3S
Contributing Partners	DFKI
Work Packages	WP3 , WP9, WP10 (requirements and interfaces)
Reference Deliverables	D3.2, D3.3, D3.4
Current Status	Under development
Short description and role	This component is managing the forgetting process. It computes the Preservation Value (PV) and Memory Buoyancy (MB) for resources based on information provided by the active system for this computation such as usage information, context information and creator as well as based on the previous MB and PV values, statistics and defined strategies and rules. The results of the Forgettor Component will be made accessible for the active system via the *Metadata Repository* (see shared components). It is planned that the Forgettor is activated on a regular basis. For this purpose it interacts with the *Scheduler*.
Delivery Mode	REST service
Subcomponents	(1) *Assessor*: The Assessor calculates values for current information value assessment of a resource, especially MB and PV. It takes into account different forgetting strategies as well as the previous values and statistics from the last computation. (2) *Strategy Manager* (database): Different forgetting strategies and rules are managed here. (3) *Statistics/Value Repository* (database): Storing all the values computed for information value assessment (including MB and PV) and statistics and using it for computing new values. (4) *Analyzer*: Classifying resources based on a strategy and the values computed in Assessor and statistics.
Main APIs, input and output formats	INPUT: resource IDs and metadata associated with the resource (mainly information on resource usage and resource context). The metadata should be a flexible data-structure, e.g. key-value pairs, so that it can be extended for different cases. OUTPUT: Computed MB and PV values together with classification in more high level classes (e.g. ”to be preserved” or ”low importance”); these will be used to update the respective information in the Metadata repository to make the most current values available for the active system. It is still under discussion if the Forgettor also informs the scheduler or the active system about the completion of the computation (e.g. via event listeners).
Plan for integration at M18	A first simple version of the Forgettor with a simple function for computing memory buoyancy
Plan for integration at M27/M36	Intermediate and final release of the overall Forgettor
Language, runtime framework	Java
SW and HW Requirements	Data base for storing historical values
Dataset for testing	usage logs and other usage information from the applications
License	Open Source

Table 7: Forgettor (Middleware, WP3 component)

Component Name	**Extractor**
Partner Responsible	CERTH
Contributing Partners	USFD, TT
Work Packages	WP4
Deliverables	D4.2, D4.3, D4.4
Current Status	Under development
Short description and role	The Extractor will take as input the original media items (e.g. a text, a collection of texts, or a collection of images) and extract information that is potentially useful not only for the subsequent execution of the Condensator, but also for other components or functionalities of the overall ForgetIT system (e.g. search). This extracted information will constitute the Extractor component's output, and will be provided in simple text or XML files (to be decided, depending on what is most convenient for integration; there is some flexibility). We envisage the Extractor to include subcomponents that will perform the following tasks:(1) Named entity extraction from text, (2) Tokenization, (3) Visual feature extraction from images, (4) Concept detection in images, (5) Image visual quality assessment. The Extractor could be either a command line tool or a REST service (both options seem feasible). A first release of at least some of the Extractor's subcomponents will be available on M12 (as part of D4.2), and a first release of the complete Extractor will be available on M18 (D8.3 deadline)
Delivery Mode	Command line tool or REST service
Subcomponents	(1) Named entity extraction from text, (2) Tokenization, (3) Visual feature extraction from images, (4) Concept detection in images, (5) Image visual quality assessment
Main APIs, input and output formats	Input: One text file or a collection of text files or a collection of images. Output: plain text or XML files with analysis results
Plan for integration at M18	A first release of the overall Extractor, integrating most of its subcomponents
Plan for integration at M27/M36	Intermediate and final release of the overall Extractor
Language, runtime framework	Matlab/Octave, C++, Executables/binaries which require OpenCV libraries (.dll/.so)
SW and HW Requirements	Operating System: Windows or Linux, Matlab/octave, GPU: NVIDIA (desired)
Dataset for testing	(1) travel of two colleagues to Edinburgh 2013, (2) travel pictures from CostaRica2013, (3) A dataset of 1000 images has been assembled and experiments are being run for blur detection, (4) Other external datasets (e.g. TRECVID)
License	OpenCV - BSD license (GPU_SURF and SURF implementation, an application of Surf algorithm is patented in the US), liblinear - Copyright (c) 2007-2013 The LIBLINEAR Project.

Table 8: Extractor (Middleware, WP4 Component)

Component Name	**Condensator**
Partner Responsible	CERTH
Contributing Partners	USFD, TT
Work Packages	WP4
Reference Deliverables	D4.2, D4.3, D4.4
Current Status	Under development
Short description and role	The Condensator will get as input the Extractor's output and possibly also the original media items that were processed by the latter in order to generate this output (or a subset of these media items). Based on this input, the Condensator will perform further text and image analysis tasks whose results are specific to the condensation process and thus of no need to other parts of the ForgetIT system, and will use all the available analysis results for performing text and image collection condensation. No feedback loop from the Condensator back to the Extractor is foreseen (thus, the Condensator can only be called after the Extractor has been executed for the same data, and the Condensator's results will not be fed in any way back to the Extractor). The final output of the Condensator will be the condensed (i.e., summarized) media items or collections, or pointers to them (depending on media item modality and on what is more convenient for integration, to be decided at a later stage). Any other analysis results generated within the Condensator for the purpose of supporting the generation of the condensed media collections most probably will not be returned to the system (since, by definition, these are only intermediate results useful for condensation; otherwise, their extraction would be part of the Extractor). We envisage the Condensator to include subcomponents that will perform the following tasks: (1) *deeper* linguistic analysis, (2) Text summarization, (3) Face detection and clustering, (4) Image collection summarization. The Condensator could be either a command line tool or a REST service (both options seem feasible). A first release of at least some of the Condensator's subcomponents will be available on M12 (as part of D4.2), and a first release of the overall Condensator will be available on M18 (D8.3 deadline)
Delivery Mode	Command line tool or REST service
Subcomponents	(1) *deeper* linguistic analysis, (2) text summarization, (3) face detection and clustering, (4) image collection summarization
Main APIs, input and output formats	Input: the output of the Extractor, which is plain text or XML files with analysis results, and the original text and image items, Output: text files, image files and plain text or XML files with analysis results
Plan for integration at M18	A first release of the overall Condensator, integrating most of its subcomponents
Plan for integration at M27/M36	Intermediate and final release of the overall Condensator
Language, runtime framework	Matlab/octave, c++ Executables/binaries which require OpenCV libraries (.dll/.so)
SW and HW Requirements	Operating System: Windows or Linux, Matlab/Octave
Dataset for testing	(1) travel of two colleagues to Edinburgh 2013, (2) travel pictures from CostaRica2013, (3) Other external datasets (e.g. TRECVID)
License	OpenCV - BSD license

Table 9: Condensator (Middleware, WP4 Component)

Component Name	**Contextualizer**
Partner Responsible	USFD
Contributing Partners	USFD, L3S, CERTH
Work Package	WP6
Deliverables	D6.1
Current Status	In development
Short description and role	The Contextualizer will embrace different subfunctionalities including as its core functionality the equipment of information objects with sufficient context information for their long-term interpretation and use, taking as input the original media items (e.g. images, text, etc.) as well as the output of the Extractor and Condensator. Furthermore, it might also use external data sources for enriching the context information (e.g. Wikipedia, or other pictures for the same event). This information will be used to determine the context required to unambiguously describe the input media. This context is likely to be defined with reference to an ontology; either large public ontologies, such as DBpedia, Freebase etc., or private ontologies from the PIMO etc. Storing a complete copy of an ontology with each preservation package is likely to be highly inefficient, and so this component will also be responsible for determining the minimum context that can be stored without loss of information. It will also interact with the Collector/Archiver for preparing the context information for packaging. Contextualization will be triggered by the intend to archive an individual information object or a set of information objects. The exact nature of the contextualizer is likely to differ dependent upon the media type. The exact formatting of the context has yet to be formalized although most tools will output XML encoded data. Furthermore, it will also be responsible for reflecting evolution in the active system (and the world) into the stored context information, in order to keep the information objects as well as the context information useful and understandable on the long run. This requires mechanisms to get informed about major changes in the active systems (e.g. in the ontology). Furthermore, it has to be identified which stored contexts are effected by these changes and change has to be represented and propagated into the Archive. For bringing information objects back into active use, a mechanism is required to apply the encoded change to the context information and/or to use them for integrating the information object into the current context (re-contextualization). This might also require using external resources such as Wikipedia or organizational information, if the captured context information is not sufficient. A further type of change that can effect the understandability is terminology evolution, which has to be detected and reflected in the context, in order to keep things findable.
Delivery Mode	REST service, initially a command line tool
Subcomponents	different components for text versus images etc., components for contextualization and re-contextualization
Main APIs, input and output formats	REST service, definition of interfaces and response formats is in progress
Plan for integration at M18	Basic component (separate components for different media types)
Plan for integration at M27/M36	A more integrated component with advanced functionality
Language, runtime framework	Java, C++
SW and HW Requirements	Linux
Dataset for testing	Not decided yet, probably user experiments
License	released by USFD under LGPL

Table 10: Contextualizer (Middleware, WP6 Component)

Component Name	**Navigator**
Partner Responsible	USFD
Contributing Partners	EURIX, L3S
Work Package	WP6, WP8
Reference Deliverables	D8.2
Current Status	Started component design
Short description and role	It is likely that the Navigator component will be highly integrated into the use case tools and as yet the form it will take has not been fully defined. The component will, however, be responsible for allowing users to see preserved items in their context (both the context at the time of preservation and at retrieval) and allow the navigation of the archive via links to other items via shared context links (i.e. a photo collection of a trip to Edinburgh in 2013 would share a context with a diary about a trip to Edinburgh in 2016). It is envisaged that an initial version of this component will focus on providing a search interface across both active and archived information, with later versions incorporating more ideas around context navigation. Such a component would require access to both active and preserved content and may need to be deployed within use case tools and the archive (as a storlet) as well as within the middle layer to make navigation feasible within sensible time constraints. This requires support for a shared index or a method for combining the results from two indices into a meaningful way. For the ranking, a time-aware search support is required, which favors information objects in active use over information objects from the archive.
Delivery Mode	REST Service
Subcomponents	Index management, time-aware search support, context navigation support
Main APIs, input and output formats	REST service, definition of interfaces and response formats is in progress
Plan for integration at M18	A basic component
Plan for integration at M27/M36	Advanced features depending on implemented scenarios
Language, runtime framework	Not decided yet
SW and HW Requirements	Linux
Dataset for testing	Not specified yet, probably user experiments
License	Released by USFD under LGPL

Table 11: Navigator (Middleware, WP6 Component)

Component Name	**Collector/Archiver**
Partner Responsible	LTU
Contributing Partners	LTU, dkd, DFKI, EURIX
Work Package	WP5
Reference Deliverables	D5.2, D5.3, D5.4
Current Status	Under development
Short description and role	Triggered by an event from PoF, this component contacts the active systems and collects data objects that should be preserved and prepares and submits them to the preservation system by packaging the data objects together with relevant metadata (into a SIP). The component receives a preservation request. Acting on that request, the Collector fetches the object and metadata from provided reference, via the CMIS interface of the active system. The Collector notifies the ID Manager with the CMIS-Id (GUID) of the object, and notifies the PoF Bus that an object has been collected. Before a Submission Information Package (SIP) can be created, other PoF components, such as the Extractor and Condensator, need to process the object and extract relevant metadata and other characteristics needed for the forgetting process. This metadata is stored in the Metadata Repository, and on a trigger from the bus, the Archiver fetches metadata and prepares the package and submits it to the Preservation System. There are at least two options here: 1. The Archiver sends a reference to where the Preservation System can fetch the package; 2. The Archiver sends the package to the ingest folder of the Preservation System. In response to the submission, the Archiver needs an ”archive ID” that should be sent to the ID Manager. The Collector/Archiver is also responsible for restructuring Dissemination Information Packages (DIP) into packages that the active system can handle to get the information back into active use. As a response to a trigger, that can come from the active system, or from PoF internal components (e.g. the scheduler), a request is made to the Preservation System for a DIP. The DIP is then disassembled and restructured if needed for adoption in the active system. This may include restructuring of metadata in order to facilitate ingest into the active system. Transformation of content objects is not considered to be a part of this functional entity. Communication will chiefly be with active systems, and with the Preservation system.
Delivery Mode	Command line tool or REST service
Subcomponents	Packager, Metadata extractor, Hash generator/checker
Main APIs, input and output formats	Input/output: REST, REST-AtomPub, CMIS, custom PoF XML schema, TAR package (SIP/DIP)
Plan for integration at M18	Ingest workflow in place with basic functionality; Simple Re-contextualization of DIP into active system
Plan for integration at M27/M36	Continued development of ingest workflow, and in particular Re-contextualization. Final component available at M36
Language, runtime framework	Java/J2EE, Java App Server/Tomcat
SW and HW Requirements	Required Linux OS
Dataset for testing	Example data from applications
License	Not yet available, possibly Open Source
Notes	Dependencies: Condensator, Extractor, Contextualizer (i.e. Metadata Repository) for metadata. Communicates with PoF adapters in active systems and AIS.

Table 12: Collector/Archiver (Middleware, WP5 Component)

3.4 OAIS Platform and Cloud Storage

The Archive, which is compliant to the OAIS model, is described in Table 13. The cloud storage system is described in Table 14. Together they realize the functionality for long term storage.

Component Name	**OAIS Platform**
Partner Responsible	EURIX
Contributing Partners	
Work Package	WP8
Reference Deliverables	D8.3, D8.4, D8.6
Current Status	Stable implementation available, to be customized
Short description and role	The Archive is compliant to OAIS model [9] and implements the main functional blocks: *Ingest*, *Access*, *Preservation Planning*, *Data Management*, *Administration* and *Archival Storage*. Several existing solutions have been evaluated in order to choose a candidate implementation, since creating a new OAIS preservation platform is beyond the scope of the project. The candidate solutions have been selected using different criteria (see Section 5). The Archive should support the packaging model selected in the project, each AIP will have its own identifier which is assigned by the archive. Other identifiers could also be used, for example those assigned by content producer and referring to original materials (before digitization) or to other editorial categorizations (this is typically managed using multiple DublinCore identifiers). The identifiers referring to the content in the active systems should be managed by other Middleware components, such as the ID Manager. The archive provides different mechanism for importing and exporting packages, including a REST API for ingest and access. Concerning access, the supported queries will include the search by AIP identifiers or by minimal descriptive metadata (e.g. DublinCore). OAI-PMH should be supported too. The archive includes its own workflow engine for managing ingestion and access and for executing preservation rules. The customization of the archive will include the development of an adapter to integrate the PDS.
Delivery Mode	Platform running in application server on dedicated machine
Subcomponents	Ingest, Access, Data Management, Preservation Management (Preservation Planning and Administration), Archival Storage (provides interface to Cloud Storage). A generic REST interface supporting OAI-PMH should be available. For what concerns Ingest, almost all available solutions support METS [11] as main metadata format for AIP, an additional component for metadata conversion could be necessary to adapt the metadata representation coming from middleware. For the Preservation Planning, processes executed close to the data (e.g. fixity checks or format transformation) will make use of Storlets, a module for executing such processes or for logging results in the AIP must be developed. For Archival Storage, a module for storing AIPs, integrating PDS, must be implemented.
Main APIs, input and output formats	Provides REST interface for ingest and access, AIP packaging model makes use of METS for metadata description and a packaging format based on BagIt or derivatives. Supported protocols include OAI-PMH, an additional CMIS or JCR compliant interface must be implemented if required.
Plan for integration at M18	Fully implemented and integrated. The adoption of an existing solution enables an early integration of the archive in the overall architecture.
Plan for integration at M27/M36	Refinements due to possible changes in the components or to demonstrate additional scenarios.
Language, runtime framework	Java, running in application server
SW and HW Requirements	Linux server, enough disk space for resources and services (no additional storage for AIPs)
Dataset for testing	Picture dataset as first test, then content from both scenarios
License	Open Source, GPL or BSD compliant
Notes	Based on the evaluation of different candidate solutions, DSpace has been selected for implementing the Archive in the PoF architecture.

Table 13: OAIS Platform (Archive, WP8 Component)

Component Name	**Cloud Storage Services**
Partner Responsible	IBM
Contributing Partners	
Work Packages	WP7
Reference Deliverables	D7.2, D7.3, D7.4
Current Status	Under development
Short description and role	Preservation DataStores (PDS) is an OAIS-based preservation-aware storage that serves as an advanced Archival Storage and supports offloaded functionality. At the top, it provides an OAIS-based interface for operations on AIPs (e.g., ingest, access, delete), as well as an interface for preservation actions (e.g., check fixity, transform, add aggregation). At the bottom end, it utilizes various generic cloud storage and compute from different providers. In addition, the system includes a Storlet Engine that can be plugged into a private cloud or object storage to execute computation modules (called Storlets), close to the data (transformations or other resource consuming tasks executed on the archived content can be done directly in the Archive without requiring to extract it to a server and put it back into the Archive). The PDS interface specification can be found on the ENSURE project web site [12, 13]. The underlying cloud storage that we'll use in PDS for ForgetIT is the OpenStack Swift Open Source framework, enhanced with a Storlet Engine to perform computations close to the data. Building the Storlet Engine and storlets for ForgetIT data and use cases is the main focus of WP7. Examples of potential storlets for ForgetIT: (1) Summarization and aggregation processes to enable managed forgetting in the archive, (2) Redundancy detection and deletion processes to support managed forgetting, (3) Multimedia analysis algorithms, (4) Integrity checks to make sure the data is not altered over time, (5) Format transformations
Delivery Mode	REST service
Subcomponents	PDS, OpenStack Swift, Storlet Engine
Main APIs, input and output formats	PDS interfaces are documented in WP7 deliverables
Plan for integration at M18	A first release of PDS with Storlet Engine and some ForgetIT storlets
Plan for integration at M27/M36	Intermediate and final release of PDS with Storlet Engine
Language, runtime framework	Java, Python
SW and HW Requirements	Linux with OpenStack software and our extensions - we'll provide our own virtual machines
Dataset for testing	Depends on ForgetIT use cases and storlets
License	Proprietary

Table 14: Cloud Storage Services (Archival Storage, WP7 Component)

4 Architecture Diagrams and Integrated Workflows

In this Section we present a first round of more in depth modeling of the PoF architecture and the related processes. For this purpose, we adopt UML2 notations creating static and dynamic diagrams of the architecture. In more detail, we are currently using component diagrams among structure diagrams and activity diagrams among behavior diagrams. Further, more fine-granular diagramms will be exploited in the next round of modelling. In addition to deployment and component diagrams for the PoF Framework architeture, we also describe two initial priority workflows which will be used to drive and validate early integration of the main components in the PoF Middleware as well as their interaction with the active systems and the archive system. For those integrated workflows we provide activity diagrams.

4.1 Structure Diagrams

The component diagram of the architecture is depicted in Figure 4. It is based on the architecture overview presented in Figure 1 and adds main interfaces between system components. The functionality of the individual components has already been described in the previous section. In Figure 3 a deployment diagram is shown, based on a possible configuration of systems and nodes which will be implemented for development and testing (see also Section 8).

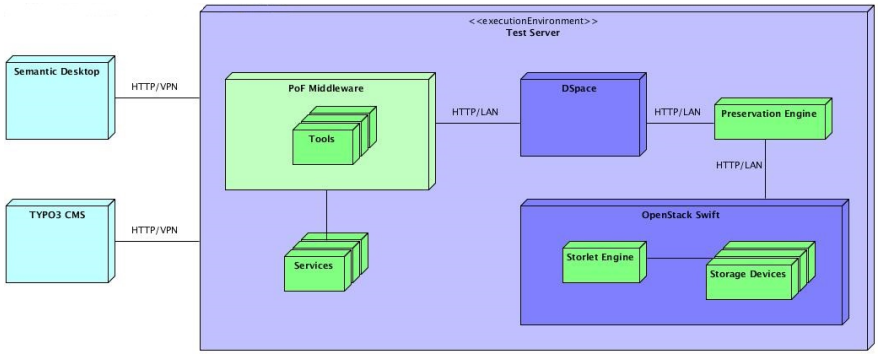

Figure 3: Deployment diagram of the PoF architecture

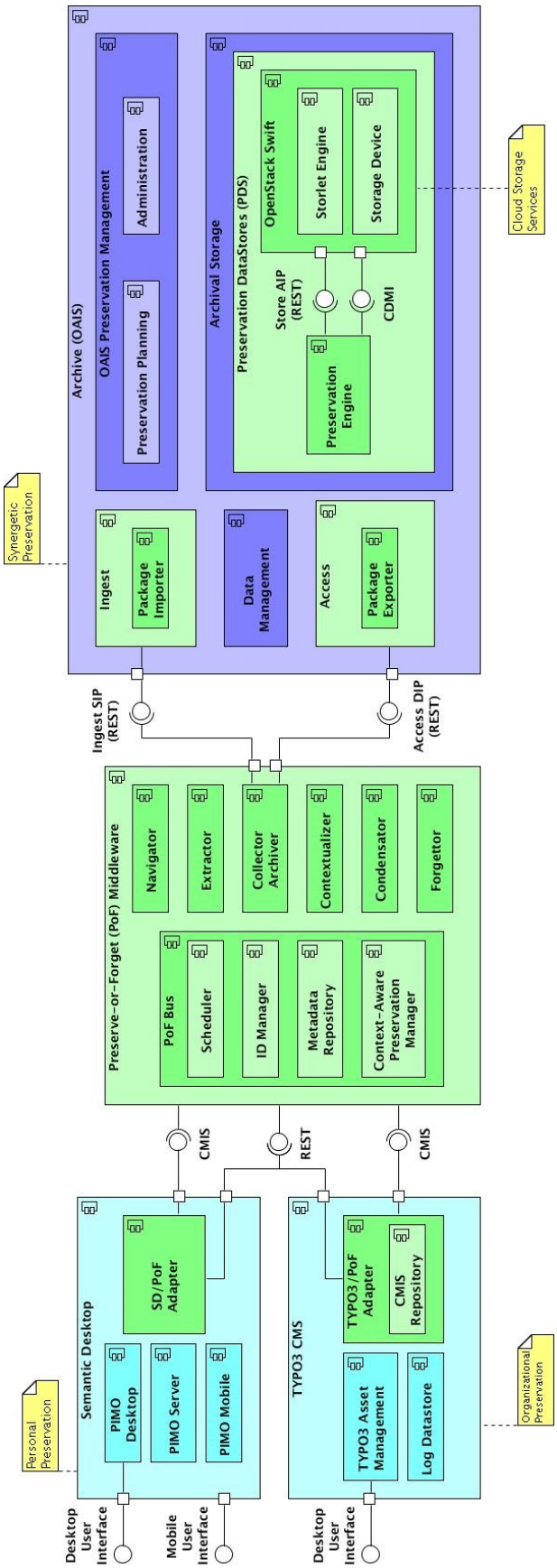

Figure 4: Component diagram of PoF architecture (*green*: components developed during the project; *blue*: existing components to be improved and customized.)

4.2 Integrated Workflows

Within the ForgetIT project it has been decided to identify a smaller set of workflows including core ForgetIT functionalities. These workflows will be used to define priorities for the integration activities in the first phase of the ForgetIT project (see also section 7). Figures 5 and 6 show the workflows that have been defined for this purpose, with the following notations: (1) complex activities are written in bold (a subdiagram will be defined in the future), while non-bold is used for actions related to less complex actions, (2) red arrows are use to distinguish an ”object flow” (red) from ”control flow” (standard case, in black).

The first workflow (see Figure 5) focuses on a basic form of synergetic preservation, which enables the smooth transition of a resource from the active system into the archive passing through the process of contextualization and packaging. The second workflow (see Figure 6) focuses on core functionality of information value assessment as it is required for realizing the Managed Forgetting process. Both processes are described in more detail below.

In implementing these first workflows the focus will be on ensuring proper interaction between the individual components in the architecture. More advanced functionality for the individual components will be added stepwise to the individual components in later phases of the ForgetIT project.

4.2.1 Workflow 1: Basic Synergetic Preservation

The workflow depicted in Figure 5 defines the process for a basic form of synergetic preservation and shows the involvement of the ForgetIT components into this process. Preservation is initiated by a preservation process, which can be scheduled either to start on a regular basis or to be triggered by an event. In both cases the Scheduler, which is part of the PoF bus (see Figure 4), will start the archiving process for a selected set of resources. This process inspects the preservation values and the preservation status that is stored for the resources in the Metadata repository and - based on this information - decides, which of the resources are handed over to preservation. In the first phase of the project, we focus on cases where the preservation value is explicitly manipulated in the active system. A change of the preservation value is triggered by a preservation request in the active system. The update of the preservation value is written into the Metadata repository.

As a next step after deciding about preservation, a simple contextualization activity is performed. This step adds core context information to the resource(s) to be archived. For this purpose, the Contextualizer is used, which interacts with the Metadata repository, the Extractor and possibly also with the active systems and external sources to collect the required context information. The core challenge here is to derive concise context information, which enables future interpretation, while not overloading the archive with unnecessary information.

Both the resource(s) to be archived and the context information are handed over to the Collector/Archivar component for packaging them into a SIP package. After this step the resource is ready to be handed over to the Archival Information System.

At this point the archival of the packages in the Archival Information System is initialized. This includes the checking of the packages for fitness to be archived. As a result of this fitness test a transformation might become necessary. In the ForgetIT system this transformation will be performed by a storlet foreseen for this purpose in the Cloud Computing Storage System, which is part of the Archival Storage employed (see Figure 4). Therefore, the transformation will be scheduled and will be performed once the packages have been transferred into the Archival Storage, which is the next step in the workflow.

If the archival is successful the ID(s) assigned to the resource(s) in the archive are returned to the PoF and stored in the ID Manager together with the time of archival, in order to enable translation of an ID of the resource in the active system to the ID of the resource in the archive. Furthermore, the scheduler will be informed about the completion of the archival action.

In case of the failure of the archiving process, the Scheduler is notified accordingly and decides about the notification of the active system.

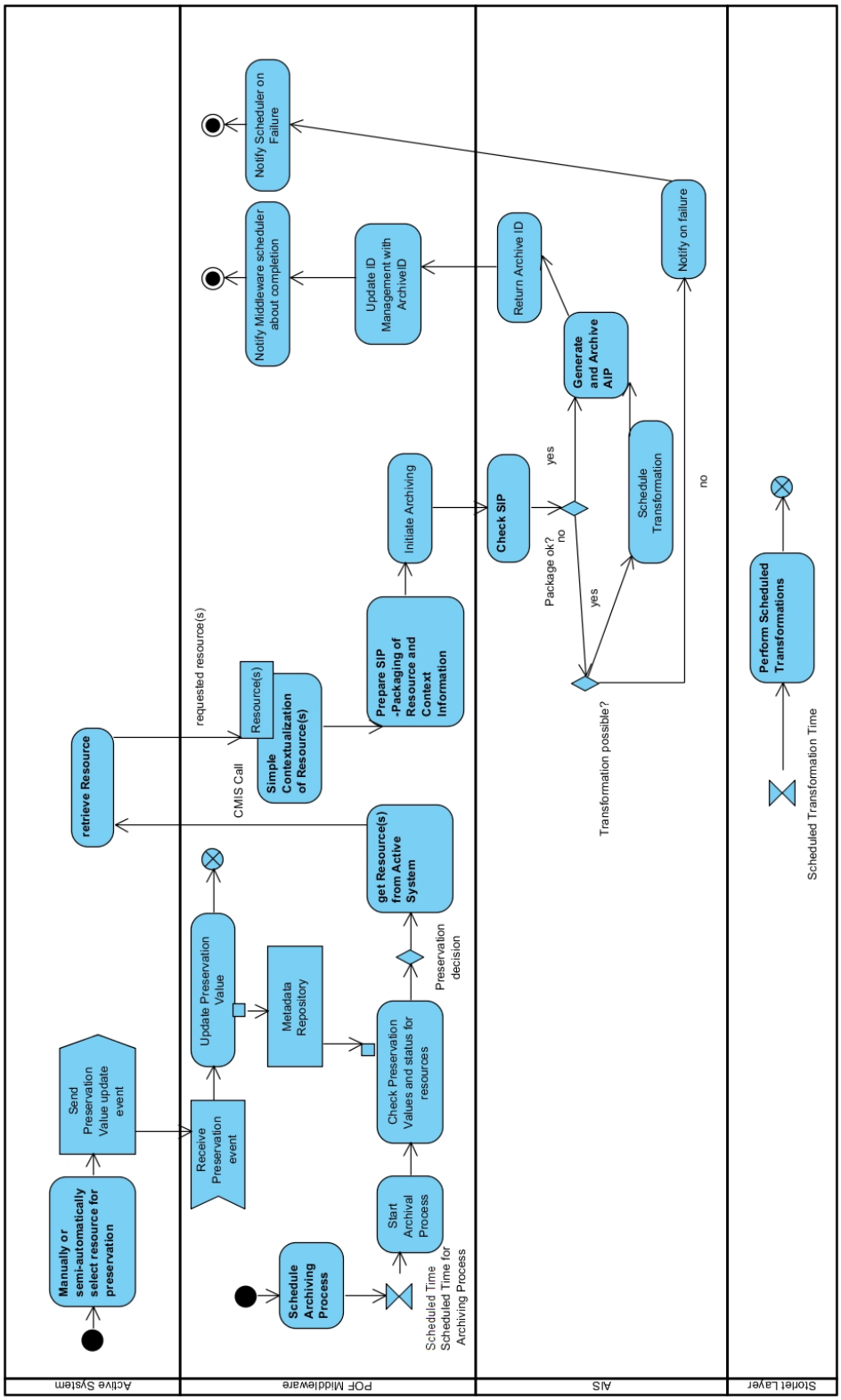

Figure 5: Priority Workflow for Basic Synergetic Preservation (complex actions in bold)

4.2.2 Workflow 2: Basic Managed Forgetting Support

The workflow Basic Managed Forgetting Support (see Figure 6) is made up of two separate processes for information value assessment, which both serve the purpose of helping the user to better structure his/her information space according to the value or importance of information (memory buoyancy). Such information value assessment is the starting point for the managed forgetting process.

The first process starts from the observation of usage of resources in the active system. Such information about the usage is collected by the active system and transferred into a Message cache on the side of the PoF Middleware. Independent from this activity a forgetting action can be scheduled in the PoF Middleware. This is done by the Scheduler, e.g., on a regular basis. Once the scheduled Forgetting action is started, the Forgettor component uses the relevant usage information from the cache (see above) and possibly also further context information to compute new values for memory buoyancy (MB) and/or preservation value (PV) for the considered resources. In addition, it also uses previous MB and PV values as well as strategies for this computation. The new and classified MB (and PV) values are stored into the Metadata Repository, which is part of the PoF bus. The new values in the Metadata Repository can subsequently be accessed by the active system, for example, in order to adapt the visualization resp. accessibility of resources in the active system according to their MB value.

The second process in this workflow focuses on a quality-based contribution to Memory Buoyancy rather than a usage-based one as in the case of the first process. This second process is started by the active system by triggering a quality assessment process for a newly captured set of images (e.g., in the Semantic Desktop) via the Scheduler. The scheduler activates a service for performing the quality checking. This quality-assessment service, which is part of the Extractor component gets access to the collection of images under consideration, e.g., by uploading them. Subsequently the automated quality assessment of the images is performed and the results of this assessment are stored into the Metadata Repository. The quality assessment results in the Metadata Repository can be accessed by the active system and can be used to trigger further activities such as suggesting some of the images for deletion.

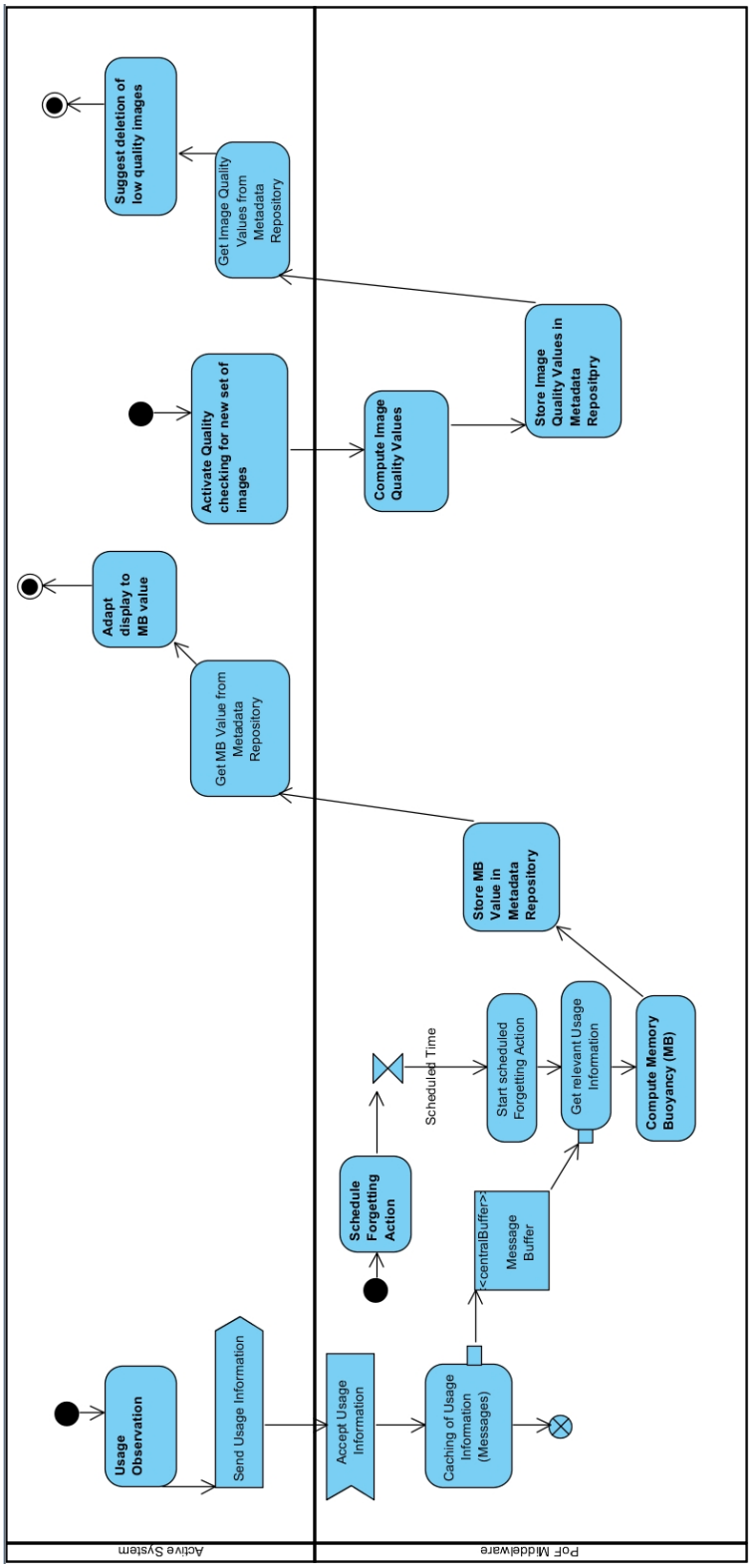

Figure 6: Priority Workflow for Information Value Assessment and Managed Forgetting (complex actions in bold; red arrows: object flows; black arrows: control flows.)

5 OAIS solutions

An Archive component will be integrated into the ForgetIT architecture. It will provide an interface to the PoF middleware and will integrate the cloud storage service.

For the actual implementation of the Archive in the PoF framework, the approach adopted in the project has been to evaluate existing solutions compliant to the OAIS model, focusing on Open Source solutions developed by other research projects and initiatives. The rational for this decision is twofold: on the one hand, since several initiatives have already developed preservation platforms inspired by OAIS supporting a large variety of content types and formats, adopting an existing solution for this component rather than reinventing the wheel is mandatory to ensure effective use of project resource: this allows focusing effort on the added values provided by ForgetIT, namely the PoF approach; on the other hand, commercial solutions available on the market have been discarded on purpose, since an Open Source and free solution rather than a proprietary implementation better suits the objectives of the project and the possibility to integrate the results developed in the technical work packages.

The preliminary evaluation of the available OAIS platforms resulted in the selection of a candidate for implementing the Archive (see Section 5.4). Further evaluation activity will be performed, based on available datasets, to implement the simple scenarios described before. The results will be part of D8.2.

5.1 Assessment criteria

The OAIS solutions reported in the following Sections provide a non-exhaustive list of Open Source platforms which have been evaluated for the implementation of the ForgetIT Archival Information System. The following requirements have been taken into account for each solution:

- Open Source license, documentation, community, last stable release
- Technology Readiness Level (TRL), e.g. prototype or stable solution
- programming language, adopted technologies, runtime environment
- APIs and protocols for integration
- archive data model and packaging
- supported content types (see D9.1 [6])
- integration of cloud storage services (see D7.1 [5]) and other components

TRL is a measure used to assess the maturity of evolving technologies such as devices, materials, components, software, work processes, etc., where the assessment is performed during its development and in some cases during early operations. Typically, when

a new technology is introduced (just invented or conceptualized), it is not suitable for immediate application and integration in broader systems. New technologies are usually subjected to experimentation, refinement, and testing. Once the technology is sufficiently proven and has been validated, it can be integrated into a system/subsystem. TRL measure originates from aerospace environment and applications (just like OAIS), but gained popularity also in other fields, such as software programming (see [14] and references therein). TRL provides a common understanding of technology status, although readiness does not necessarily fit with appropriateness or technology maturity. In the context of ForgetIT, we used this measure associated to different candidate solutions for the Archive, in order to provide summary information about the readiness of the solution for integration in the overall architecture. The assigned value is based on the evaluation of the platforms against the criteria mentioned above. Since different definitions are used and significant differences exist in terms of maturity at a given technology readiness level, we used the scale adopted by U.S Department of Defense [15] (and the similar ones from NASA and ESA, reported in [14]) and will provide a short description in each table for convenience.

5.2 Candidate platforms

For each solution a fact sheet is provided, with information relevant for the purpose of the present document, that is selecting a candidate implementation of the archive in the PoF architecture. Additional information, technical details and support are available on the project web site provided for each solution.

Disclaimer 1 Based on assessment criteria above, we identified the best candidate solution. Anyway the evaluation of these platforms is still in progress: additional feedbacks will be provided when taking into account actual integration tests with other ForgetIT components and data sets during the testbed. Even if the probability of changing candidate Archive implementation is low, the ultimate choice will be reported in D8.2.

Disclaimer 2 Commercial and proprietary solutions haven't been considered here.

The following solutions have been considered:

- DSpace, described in Table 15
- RODA, described in Table 16
- Archivematica, described in Table 17
- Fedora, described in Table 18
- P4, described in Table 19
- iRODS, described in Table 20

For each platform we provide a fact sheet with a short description of the main features and a discussion against the assessment criteria mentioned above.

Name	**DSpace**
Project Page	`http://www.dspace.org`
License	BSD
Short Description	DSpace is an out-of-the-box Open Source repository application for delivering digital content to end-users, typically used for creating open access repositories for scholarly and/or published digital content. It is considered the most widely used Open Source repository software for non-profit and commercial organisations. DSpace captures, stores, indexes, preserves and redistributes an organization's research material in digital formats. Research institutions worldwide use DSpace for a variety of digital archiving needs - from institutional repositories (IRs) to learning object repositories or electronic records management, and more. DSpace is freely available as Open Source software, which can be customized and extended. An active community of developers, researchers and users worldwide contribute their expertise to the DSpace Community. While DSpace shares some feature overlap with content management systems and document management systems, the DSpace repository software serves a specific need as a digital archives system, focused on the long-term storage, access and preservation of digital content.
Source Code, Documentation, Community	Source code available on Sourceforge [16] and GitHub [17], documentation available on project web site [18], the project community is supported by DuraSpace [19], and institutions adopting DSpace.
Last Stable Release	3.2 (July 2013)
TRL	**8** (Actual system completed and qualified through test and demonstration)
Language, Runtime	Java, application server and RDBMS required
APIs and Protocols	REST APIs, supports OAI-PMH and SWORD
Data Model and Packaging	DSpace defines a structured Data Model [20], providing a representation of digital contents in terms of collections, communities, items, and sites, associating different levels of support for objects to be preserved. AIP is a Zip file containing a METS manifest and all related content bitstreams. Each bitstream is associated with one bitstream format. Because preservation services may be an important aspect of the DSpace service, it is important to capture the specific formats of files that users submit. In DSpace, a bitstream format is a unique and consistent way to refer to a particular file format. An integral part of a bitstream format is an either implicit or explicit notion of how material in that format can be interpreted. Each bitstream format additionally has a support level, indicating how well the hosting institution is likely to be able to preserve content in the format in the future. There are three possible support levels that bitstream formats may be assigned by the hosting institution: *Supported*, *Known* or *Unsupported*. Although DSpace provides some default values for Supported, Known and Unknown formats, each institution should determine the appropriate values based on local preservation strategy, e.g. after careful consideration of costs and requirements. Each item has one qualified Dublin Core metadata record. Other metadata might be stored in an item as a serialized bitstream, but DublinCore is stored for every item for interoperability and ease of discovery. The Dublin Core may be entered by end-users as they submit content, or it might be derived from other metadata as part of an ingest process. Items can be removed from DSpace in one of two ways: They may be 'withdrawn', which means they remain in the archive but are completely hidden from view. In this case, if an end-user attempts to access the withdrawn item, they are presented with a 'tombstone,' that indicates the item has been removed. For whatever reason, an item may also be 'expunged' if necessary, in which case all traces of it are removed from the archive. Broadly speaking, DSpace holds three sorts of metadata about archived content: descriptive, administrative and structural.

Content Types, Formats, Standards	Virtually any type of file, regardless of format or extension can be stored in DSpace. In this context, ”support” for such a file is defined as being able to upload the file, and offering it for download to end users. For text formats, DSpace offers full-text indexing and searching. For image formats, DSpace offers thumbnail generation and display. DSpace explicitly supports HTML documents keeping the cross references, although with some limitations (only static content and all links must be relative). METS is the reference standard format for SIP, AIP and DIP. Packagers are software modules that translate between DSpace Item objects and a self-contained external representation, or ”package”. A Package Ingester interprets, or ingests, the package and creates an Item. A Package Disseminator writes out the contents of an Item in the package format. A package is typically an archive file such as a Zip or ”tar” file, including a manifest document which contains metadata and a description of the package contents. DSpace Simple Archive Format can be used for export and ingest. In the current version of DSpace, handles are used as internal identifiers. DSpace uses the Handle System from CNRI as the persistent identifier for each digital object. Handles are resolved to actual URLs via a resolution service. Handles in DSpace (and elsewhere) are currently implemented as HTTP URIs, but can also be modified to work with future protocols. The Handle system is also able to support existing bibliographic identifiers such as ISBN or ISSN.
OAIS Compliance	OAIS also serves as a framework for developers of digital repository software. The impact of the OAIS model on DSpace is apparent, even if DSpace has evolved independently following several requirements. Data Management functionality leverages DSpace Data Model for handling archived items, bitstreams and metadata. Ingest is implemented by a batch ingester and a web submit UI. For Archival Storage, DSpace offers two means for storing bitstreams: the first is in the file system on the server; the second is using SRB (Storage Resource Broker), a data grid management system enabling distributed storage. Both are achieved using a simple, lightweight API. SRB is a storage manager that offers unlimited storage and straightforward means to replicate (in simple terms, backup) the content on other local or remote storage resources. Concerning Access, DSpace allows end-users to discover content in a number of ways, including via external reference, such as a Handle (containing the AIP ID), searching for one or more keywords in metadata or extracted full-text, browsing though title, author, date or subject indices, with optional image thumbnails. DSpace’s indexing and search module provides a configurable Lucene-based search engine and a API which allows for indexing new content, regenerating the index, and performing searches on the entire corpus, a community, or collection. Behind the API is the Java freeware search engine Lucene. Lucene provides fielded searching, stop word removal, stemming, and the ability to incrementally add new indexed content without regenerating the entire index. Web UI enables views of different indexes and browsing. DSpace identifies two levels of digital preservation: bit preservation, and functional preservation. Bit preservation ensures that a file remains exactly the same over time (not a single bit is changed) while the physical media evolve around it. Functional preservation goes further: the file does change over time so that the material continues to be immediately usable in the same way it was originally while the digital formats (and the physical media) evolve over time.

OAIS Compliance (continued)	Some file formats can be functionally preserved using straightforward format migration. Other formats are proprietary, or for other reasons are much harder to preserve functionally. Compared to other platforms, DSpace does not include, in the vanilla installation, tools for migrating Bitstreams from one format to the other, but these can be easily written using DSpace Java API. Concerning Administration, DSpace offers also additional features such as usage and system statistics, a checksum checker and reports. The purpose of the checker is to verify that the content in a DSpace repository has not become corrupted or been tampered with. The functionality can be invoked on an ad-hoc basis from the command line, or configured via cron or similar. Options exist to support large repositories that cannot be entirely checked in one run of the tool.
Integration of ForgetIT Components	A plugin manager is provided, although deeper analysis on the source code is required. An Add-on mechanism is provided to extend DSpace with additional components. Extension and add-ons provided by the community are maintained on the project wiki. DSpace supports DuraCloud as cloud storage solution, in order to integrate PDS additional software and a plugin must be implemented.

Table 15: DSpace

Name	**RODA**
Project Page	http://www.roda-community.org/
License	GNU LGPLv3
Short Description	RODA is a complete digital repository that delivers functionality for all the main units of the OAIS reference model. The platform is maintained by KEEP SOLUTIONS [21], and is built on top of Fedora (see Table 18). RODA is based on Open Source technologies and is supported by existing standards such as the OAIS, METS, EAD and PREMIS. A plug-in and task scheduling mechanism is provided to add more functionality to the system (e.g. new preservation events, alerts, tools, etc.). The repository natively supports normalization on ingest for different file formats. RODA can be extended to comply with more file formats or better preservation action tools. Support for migration-based preservation actions is built into the system. Preservation actions and management within RODA is handled by a task scheduler. The task scheduler allows the administrator to define the set of rules that trigger specific actions, and when these should take place. Preservation actions include format conversions, checksum verifications, reporting (e.g. to automatically send SIP acceptance/rejection emails), notification events, etc. The basic services in RODA are provided by Fedora Commons, the application framework that supports RODA. These services account for elementary tasks at the Data Management and Archival Storage level. Examples of such services are: store and index a digital object, add a data stream to a Fedora object, get a data stream, purge an object, find objects and list data streams. RODA Core Services are responsible for carrying out more complex tasks such as handling the ingest workflow, querying the repository in advanced ways and carrying out administrative functions on the repository. RODA enables tight integration of systems already existing in the client institution.
Source Code, Documentation, Community	Source code available on GitHub [22], documentation available on project web site [23], community is supported by KEEP SOLUTIONS [21] and institutions adopting RODA.
Last Stable Release	1.1.0 (July 2013)
TRL	**6** (System/subsystem model or prototype demonstration in a relevant environment)
Language, Runtime	Java, requires application server and RDBMS. Built on top of Fedora.
APIs and Protocols	REST and SOAP APIs, supports OAI-PMH
Data Model and Packaging	RODA's content model is atomistic and very much PREMIS-oriented. Each intellectual entity is described by an EAD-component metadata record. These records are organized hierarchically in order to constitute a full archival description but are kept separately within the Fedora content model. Relationships between EAD-components are created using Fedoras own RDF linking mechanism. Additionally, each leaf record (i.e. a file or an item) is linked to a representation object, i.e. a Fedora object that embeds all the files and bitstreams that compose the digital representation. Finally, each of these objects are linked together by a set of PREMIS entities that maintain information about the digital objects provenance and history of events (PO nodes). Each preservation event that takes place inside the repository is recorded as a new preservation-event node. Special events, like format migrations, establish relationships between two preservation-representation nodes. These are called linking events. Each preservation event is executed by an agent, whether this be a system user or an automatically triggered software application. The agent that triggered the event is recorded in PO agent nodes.

Content Types, Formats, Standards	RODA is capable of ingesting and normalizing (according to the preservation plan in place) text documents, raster images, relational databases, video, and audio. A plug-in mechanism enables RODA to support additional formats. RODA follows open standards using EAD for description metadata, PREMIS for preservation metadata, METS for structural metadata, and several standards for technical metadata (e.g. NISO Z39.87 for digital still images).
OAIS Compliance	RODA is composed of several functional modules supporting processes of a common archival information system. Ingest is composed by a configurable multi-step workflow that validates submitted information and also extracts technical metadata from ingested files. The ingest process also normalizes formats according to the preservation policy in place and includes both automatic and human quality assurance steps. RODA supports the ingest of new digital material as well as associated metadata in 4 distinct ways: (1) online submission (self-archiving), (2) off-line submission using an client application, (3) batch import, and (4) integration with third-party document management software via invocation of SOAP Services or client API. SIPs are submitted to a series of tests to assess their integrity, completeness and conformity to the ingest policy. After decompressing the SIP, the validation process performs different tasks, such as virus check, METS envelope syntax check, SIP completeness check, file integrity check, descriptive metadata check, preservation metadata check, representation check (at least one representation exists within the SIP) and normalization. Representations whose format do not conform to the preservation formats defined by the preservation policy are automatically converted to the correct format. The original representation is maintained by the repository. Descriptive metadata is based on the International Standard for Archival Description (ISADg) and is supported by the EAD/XML standard. RODA fully implements a configurable ingest workflow that not only validate SIPs, but also enables manual appraisal by data management professionals. Digital objects are migrated to preservation formats during the ingest process according to the policies in place. Preservation management within RODA is handled by scheduled events. The set of rules that trigger specific preservation actions and when these should take place are defined. Preservation actions comply to a common API, to enable creation and installion of new preservation actions in the repository. Preservation actions include format converters, checksum verifications, reporting tools (e.g. to automatically send SIP acceptance/rejection emails), etc. As a fallback strategy, the system always retains the original versions of digital representations, so that an emulation preservation strategy still remains viable in the future. RODA implements preservation planning through the possibility of running and scheduling preservation actions right in the administration module. Administration components allow editing of the descriptive metadata and definition of rules for preservation interventions such as scheduling integrity checks, initiate a format migration processes, or control the users or groups that are authorized to perform certain actions in the repository. RODA includes administration features such as user management, reporting, ingest workflow configuration, log viewer, permissions management, etc. Quality assurance and preservation metadata ensure authenticity of records while providing traceable records of all changes and events that occur to a digital representation. All actions performed in the repository are logged for security and accountability reasons.

OAIS Compliance (continued)	Access to data is provided through embedded web viewers and downloads. Several versions of the same data are provided, including the originally ingested digital representation. The consumer is able to browse over available collections to view or download digital representations kept in the repository. Depending on the type of the digital object, different viewers or disseminators are used. For example, text documents are delivered to consumers without resorting to any particular artifacts. They are delivered in PDF format, so the consumer should use its favourite PDF viewing application. Documents composed of several images (such as digitised works) on the other hand are displayed in special Web viewing applications that allow consumers to navigate through the pages of the representation. Data storage is managed by Fedora Commons, the data layer backend. Data is stored on the file system separately from the metadata.
Integration of ForgetIT Components	RODA exposes all its functionality via Web Services. Java APIs are available to integrate external components programmatically. Integration with cloud storage requires customization of the underlying Fedora services.

Table 16: RODA

Name	**Archivematica**
Project Page	`https://www.archivematica.org`
License	GNU AGPL v3
Short Description	Archivematica is a free and Open Source digital preservation system that is designed to maintain standards-based, long-term access to collections of digital objects. Archivematica uses a micro-services design pattern to provide an integrated suite of software tools that allows users to process digital objects from ingest to access in compliance with the ISO-OAIS functional model. Users monitor and control the micro-services via a web-based dashboard. Archivematica uses METS, PREMIS, Dublin Core and other best practice metadata standards. Archivematica implements format policies based on an analysis of the significant characteristics of file formats. Archivematica is maintained by Artefactual Systems [24], in collaboration with UNESCO and other institutions.
Source Code, Documentation, Community	Source code available on GitHub [25], documentation on Archivematica wiki [26], community supported by Artefactual Systems [24]
Last Stable Release	0.10 (April 2013), 1.0 announced (Fall 2013)
TRL	**7** (System prototype demonstration in an operational environment)
Language, Runtime	Python for implementing micro-services, requires Django MVC framework. Virtual Appliance provided for different virtualization environments (VirtualBox, VMWare, KVM)
APIs and Protocols	REST APIs, default access system is AtoM. Provides export to DSpace format. Programmatic access to indexed AIP is available through Elasticsearch [27].
Data Model and Packaging	SIP based on METS, normalization process during ingestion. LoC BagIt format (zip) used for AIP. Export to DSpace data model is supported, Archivematica can act as dark-archive for DSpace, providing back-end preservation functionality while DSpace remains the user deposit and access system. Archivematica supports also DIP upload to AtoM and CONTENTdm services.
Content Types, Formats, Standards	METS supported for ingest and access. PREMIS and DC are supported standards for preservation and descriptive metadata. Tested with documents, pictures and videos. Defines access and preservation formats for each media type and includes normalization tools (mainly ffmpeg).
OAIS Compliance	Archivematica implements a micro-service approach to digital preservation. The Archivematica micro-services are granular system tasks which operate on a conceptual entity that is equivalent to an OAIS information package. The physical structure of an information package will include files, checksums, logs, submission documentation, XML metadata, and others. These information packages are processed using a series of micro-services. Micro-services are provided by a combination of Archivematica Python scripts and one or more of the free, Open Source software tools bundled in the Archivematica system. Each micro-service results in a success or error state and the information package is processed accordingly by the next micro-service. There are a variety of mechanisms used to connect the various micro-services together into complex, custom workflows. Preservation plans available for different media types, based on analysis of the significant characteristics of the files. The user dashboard provides interface mapped onto OAIS functional entities. The web dashboard allow users to process, monitor and control the Archivematica workflow processes. It is developed using Python-based Django MVC framework. The Dashboard provides a multi-user interface that will report on the status of system events and make it simpler to control and trigger specific micro-services. This interface allows users to easily add or edit metadata, coordinate AIP and DIP storage and provide preservation planning information.

OAIS Compliance (continued)	Archivematica maintains the original format of all ingested files to support migration and emulation strategies. However, the primary preservation strategy is to normalize files to preservation and access formats upon ingest. Normalizing is the process of converting ingested digital objects to preservation and/or access formats. In Archivematica the original objects are always kept along with their normalized versions. Archivematica groups file formats into format policies (e.g. text, audio, video, raster image, vector image, etc.). Archivematica's preservation formats must all be open standards. Additionally, the choice of formats is based on community best practices, availability of free and Open Source normalization tools, and an analysis of the significant characteristics for each media type. The choice of access formats is based largely on the ubiquity of web-based viewers for the file format. Not all files can be normalized on ingest because for example there are no available Linux-based Open Source tools to handle the conversions and/or no agreed upon preservation formats. In addition, some filetypes are not necessarily in the best preservation format but are still so ubiquitous and well-supported that they need not be normalized at the present time. In these cases, the files are kept in their original formats. A Format Policy Registry is available to implement rules of Preservation Planning.
Integration of ForgetIT Components	Archivematica provides a full-fledged preservation platform which can be installed and used out-of-the-box. Extending Archivematica for integration with external components requires modification of the source code. Default storage mechanism is local file system.

Table 17: Archivematica

Name	**Fedora**
Project Page	http://www.fedora-commons.org/
License	Apache License, v2.0
Short Description	Fedora is a digital repository, developed and maintained under the stewardship of the not-for-profit organization DuraSpace [19]. The Fedora Repository Project provides a robust Open Source software system based on a core repository service (exposed as web-based services with well-defined APIs) and an array of supporting services and applications including search, messaging and administrative clients. Fedora aims at ensuring that digital content is durable by providing features that support digital preservation. The FedoraCommons refers to the community surrounding the Fedora Repository Project.
Source Code, Documentation, Community	Source code available on GitHub [28], documentation on Fedora Commons wiki [29], project community supported by DuraSpace, a registry of institutions adopting Fedora is maintained.
Last Stable Release	3.7.0 (Sept. 2013)
TRL	**8** (Actual system completed and qualified through test and demonstration)
Language, Runtime	Java, requires application server and RDBMS
APIs and Protocols	REST and SOAP APIs, supports OAI-PMH
Data Model and Packaging	Fedora Digital Object Model [30], digital objects stored internally using FOXML.
Content Types, Formats, Standards	Fedora Digital Object Model supports videos, images documents and others. FOXML format is preferred schema for ingest and access, METS (using Fedora extension) supported for ingest and access, also MPEG-21 DIDL.
OAIS Compliance	Ingest and Access available thorugh REST or SOAP APIs or Web UI. Batch ingestion supported. Supported SIP formats are FOXML or METS. Export formats are FOXML, METS and ATOM. Export to new archive or purging existing object are supported. Search possible using Web UI or REST APIs, using identifier or DC metadata. Data Management based on own object model, specific component for archive Administration is available. Archival Storage implemented by Low Level Storage interface, supporting disks and cloud services (experimental). Periodic activities for Preservation Planning are supported at the datastream level (e.g. checksums). Datastreams can be updated or migrated. Descriptive metadata can be modified, too.
Integration of ForgetIT Components	A plugin or adapter to integrate with PDS is required. DuraSpace provide their own cloud solution (DuraCloud), but it is not free. Integration of other components delivered as REST services or command line tools is possible.

Table 18: Fedora

Name	**P4**
Project Page	`http://prestoprime.eurixgroup.com/p4`
License	GPLv3
Short Description	P4 is the preservation platform developed by EU FP7 PrestoPRIME project [31]. P4 implements the main functional entities of the OAIS model for an archive managing AV content and is made up of three main components: (1) core libraries, implementing OAIS components for storage, metadata management, ingest, access, administration and preservation actions; (2) web server, providing REST interfaces for interacting with the archive; (3) web UI, providing ingest, access and administrative functionalities according to the user profile. The web server provides interfaces for ingest, access and administration. The user can ingest SIP files into the platform, get information about the status of the submitted jobs and of the whole system, search for AIP available in the archive, and get access to the DIP, through the web interface. The user interface manages local users and can connect to multiple P4 instances with different user identifiers, each associated to a specific role (consumer, producer, administrator) for that platform. The external tools and services can be integrated using a plugin framework, the motivations for this being twofold: on one hand it provides a flexible way to integrate new components (e.g. to execute some specific steps during ingestion), on the other hand the platform and the core components are decoupled from specific tools or scenarios and P4 users have access to an open framework which can be used out-of-the-box, by configuring a minimum set of parameters. P4 includes a workflow engine, a lightweight execution environment to configure custom tasks based on external tools and services, exploiting the APIs of core modules. The external tools used to implement a specific workflow can be deployed within a P4 plugin. Tools developed within the project and integrated in P4 cover metadata extraction (e.g. MXF tools), quality assessment, storage (disks via NFS or CIFS, LTO tapes, shared or federated storage systems such as iRODS and MServe), emulation (Multivalent), SLA and monitoring, rights, search and indexing (Solr), AV material segmentation and access, format migration, fixity checks. Concerning the storage configuration, different workflows have been tested in PrestoPRIME. In particular the configuration with two copies of the master quality file was implemented either with LTO tapes (two copies on two different tapes) or with iRODS as policy-driven storage (the automatic replica rule, with periodic fixity checks was defined).
Source Code, Documentation, Community	Source code available on GitHub [32], documentation available on the web site [33], currently the platform is part of the PrestoCentre tools library [34]
Last Stable Release	2.2.0 (Dec. 2012)
TRL	**6** (System/subsystem model or prototype demonstration in a relevant environment)
Language, Runtime	Java, Servlet Container required, no RDMS (XML DB)
APIs and Protocols	REST APIs, supports OAI-PMH.

Data Model and Packaging	The data model makes use of METS as the main wrapper format for descriptive and technical metadata, as well as for mapping AV resources within the AIP. Other metadata standards are supported, such as MPEG-7 for technical metadata, PREMIS for preservation events, MPEG-21 for rights representation, DublinCore for descriptive metadata and others. P4 also supports DNX, a metadata format built on top of PREMIS vocabulary, used in Rosetta. Using P4 plugins, virtually any metadata standard can be used in the AIP. Access interface supports also OAI-PMH protocol. The data model is tailored to broadcast environment (editorial entities, master and browsing qualities, B2B contracts). No compressed formats such as zip, BagIt or tarball used for AIP, METS contains references to metadata and AV files.
Content Types, Formats, Standards	Focus on videos, but other content types can be supported defining new workflows. Based on METS, supports DC, MPEG-21 CEL, MPEG-7 AVDP, DNX, PREMIS.
OAIS Compliance	Ingest and access provided by web UI or REST APIs, using METS as unique format for all OAIS information packages, common to other platforms. An advanced search engine based on Solr allows indexing of different descriptive and technical metadata. Several solutions are available for Archival Storage, supporting local and distributed storage. Preservation Planning is provided by integrated tools for fixity checks or format migration, no scheduler is implemented in the platform, makes use of external systems (e.g. iRODS). The index is stored in a fast native-XML DB and periodic triggers are executed for backup and integrity checks of the AIP XML files. Additional preservation operation are provided by storage solutions (e.g. the LTO component). Data Management and Administration are provided by the P4 web UI, including monitoring of jobs and workflows.
Integration of ForgetIT Components	The favourite integration mechanism is making use of REST interfaces over HTTP, to get loose coupling and reduce dependencies. P4 provides a plug-in mechanism to integrate external components or services in the workflow. In order to integrate cloud services, a new storage plugin should be implemented and added to the storage layer (if we use REST APIs this should be straightforward).

Table 19: P4

Name	**iRODS**
Project Page	`https://www.irods.org`
License	BSD
Short Description	iRODS is the integrated Rule-Oriented Data-management System, a community-driven, Open Source, data grid software solution. It is a policy-based data management system, implementing a micro-services pattern, based on rule engine. iRODS helps manage (organize, share, protect, and preserve) large sets of computer files. Collections can range in size from moderate to a hundred million files or more totaling petabytes of data. The requirements to manage large collections of data include both a number of generic capabilities and diverse features that depend on the details of different applications. iRODS is also highly configurable and easily extensible for a very wide range of use cases through user-defined micro-services, without having to modify core code. iRODS is used by many projects and teams, small and large, national and international, computer technologists and non. iRODS includes a set of features that blend together well and augment each other to form a comprehensive whole. iRODS major features include high-performance network data transfer and a unified view of disparate data. iRODS uses unique logical names that are separate from the names as stored physically, providing a global logical name-space via the iCAT Metadata Catalog in a DBMS to keep track of the names and locations of files so users don't have to. iRODS also supports a wide range of physical storage, including Unix and Windows files systems, archival storages systems (HPSS, tapes), etc. iRODS provides easy, automated replication and backup to multiple storage devices/locations at the physical level. So, users access the files via the logical names and the system finds and gets the physical files. iRODS also manages metadata, both system (automatic) and user-defined, and stored in the iCAT Metadata Catalog running in a DBMS. Users can query the system to find, use, verify, etc. files with particular attributes (metadata). iRODS provides fine-grained controlled access, by user or group. iRODS innovative Rule Engine applies local and community policies expressed as rules and executed via server-side micro-services. Rules invoke other rules and/or micro-services making the system highly configurable for site-specific needs and automated for cost-effective administration of today's mushrooming data collections. Workflows can be executed as part of normal operation (e.g. a Rule can be run as a file is initially stored to automatically make an offsite replica) or as delayed or periodic Rules. iRODS can operate as a complete stand-alone system (utilizing storage systems, database systems, and networks underneath) and also as middleware where higher-level and application-specific software makes use of iRODS as part of its infrastructure.
Source Code, Documentation, Community	Source code and documentation available on the project wiki [35], supported by the DICE group of the University of North Carolina at Chapel Hill and the University of California San Diego.
Last Stable Release	3.3 (July 2013)
TRL	**8** (Actual system completed and qualified through test and demonstration)
Language, Runtime	C, Perl, Shell. Provides a service running the catalog, other nodes can be distributed, requires a RDBMS.
APIs and Protocols	iRODS provides GUI, Web, WebDAV, command line interfaces, as customized shell commands used for managing content and administering the archive, and also a Java API (Jargon) allowing programmatic integration in external systems. Development of rules for specific tasks requires a custom language.

Data Model and Packaging	Data Virtualization is the underlying idea in the iRODS data grid system. Instead of a physical naming, iRODS adopts a virtual (or logical name) for every entity that interface with user or application. The mapping from the logical name to physical name is maintained persistently in the Metadata Catalog and the mapping is done at run time by the Virtualization sub-system. The virtualization pervades all aspects of iRODS and is seamlessly integrated into the various modules. iRODS provides different types of data transfer mechanism. iRODS can get and put files from a remote storage system (which is fronted by an iRODS server) or can transfer from one storage to another (as a third-party transfer). The access of the files can be either as a single file transfer or as a whole collection/sub-collection transfer. It can transfer these files in bulk mode (when several small files are being transferred) or in parallel mode when a large file is being transferred. All these different options are optimally selected depending upon the file sizes and the number of files being transferred. iRODS data model defines logical name spaces for files (POSIX, Grid and collection attributes), users, resources, rules, micro-services and states. A resource, or storage resource, in iRODS terminology, is a software/hardware system that stores data. An iRODS resource is a logical mapping of a ”resource name” to a number of physical attributes that define the resource. The iRODS clients/servers can then operate on remote or local data on different types of resources through a common interface. Currently, iRODS supports 3 resource types - unix file system, HPSS, and Amazon S3.
Content Types, Formats, Standards	Any kind of file can be virtually stored. Provides support for metadata, search and other operations on the content.
OAIS Compliance	iRODS software was designed to allow curators utilising heterogeneous storage and computing facilities to define policies without being concerned with the technical detail of how the system implements those policies and without having to respond to changes in technical infrastructure. iRODS uses a data grid architecture, running server software and a rule Engine on each server that will become part of the virtual repository. A separate, unique iRODS iCAT Metadata Catalog uses a database to track descriptive and preservation metadata. Users determine workflows and automated tasks that the Rule Engine carries out regardless of the originating server. iRODS was not conceived as an implementation of the OAIS model, but to fullfil other requirements for a rule-oriented data management system leveraging grid technologies in the context or research and academic institutions. Nevertheless, iRODS is widely used in the research community, in high performance computing projects, and in preservation environments and digital libraries. Several principles borrowed from OAIS could be implemented in iRODS. For example, OAIS describes a standard model for access to information repositories that could be ported on top of iRODS. Within the iRODS data grid, standard functions (micro-services) are defined which can be composed into workflows to support procedures that are applied to the contents of the information repository. Data Management procedures are controlled by policies that are managed in a distributed rule engine.
Integration of ForgetIT Components	iRODS provides a mechanism for integrating external components as additional micro-services or rules. Additional components for ingest and access should be developed on top of iRODS, unless some of the add-ons developed by the community already fits with ForgetIT requirements. iRODS is already based on Data Grid and provides a policy-driven storage, enabling micro-services and rules running close to the data. This feature conflicts with one of the expected outcomes of the project, namely the cloud storage services, which should provide analogue features using the Storlet technology.

Table 20: iRODS

5.3 Other solutions, projects and initiatives

Several EU projects have been funded in the field of digital preservation. Almost all projects developed tools for digital preservation which have been partially or totally delivered as Open Source. The outcomes of such projects have also provided valuable feedbacks to the digital preservation community and have been often used to support the initial development of popular digital preservation systems. Typically, the main limitation of such tools is due to the short lifetime of the supporting projects, which have the primary focus of demonstrating the project objectives rather than providing stable software. Software engineering and maintenance require resources which cannot always be guaranteed after the project end. Several initiatives such as the Digital Preservation Coalition, the PrestoCentre or the The Planets Foundation (just to mention a few), try to support the community in maintaining software tools and libraries for the digital preservation.

After the initial phase where the focus of projects like DELOS [36] was to raise awareness on the digital preservation issues and different initiatives such as the Digital Curation Centre and Digital Preservation Europe started, several projects have been funded focusing on research and development of digital preservation technologies, processes, audit and other technical aspects. Such group includes CASPAR [37], PLANETS [38], Shaman [39], PrestoPRIME [31], SCAPE [40], ENSURE [12], TIMBUS [41] and many others. Some Coordinated Actions such as Presto4U [42] have also been funded. An other example of Open Source digital preservation platform is DPSP[1], a collection of software applications developed by National Archives of Australia[2], supporting the goal of digital preservation. The Digital Preservation Coalition [43] provides also technology watch and training, supporting the dissemination and adoption of best practices and technologies in the digital preservation community.

5.4 Selection of the OAIS platform

The OAIS solutions reported in Section 5.2 are good candidates for the implementation of the Archive in the PoF architecture. Taking into account the criteria identified in Section 5.1, at the moment the most promising one seems to be **DSpace**.

DSpace is stable and supported by a huge community of users and developers and has been adopted by about one thousand institutions worldwide, which have chosen it as the reference solution for their institutional repositories. The compliance to OAIS model, mainly for what concerns aspects related to Preservation Planning, is progressing but additional customization is required. Fedora, the other solution from DuraSpace, is not conceived as an out-of-the-box implementation and the amount of work required to customize and prepare the platform is still considerable. RODA, which is built on top of Fedora, provides additional features related to the actual preservation of the contents and is strongly OAIS-oriented. It is a relatively new project which could benefit from wider

[1] `http://sourceforge.net/projects/dpsp`
[2] `http://naa.gov.au`

adoption. Archivematica is compliant to OAIS and provides advanced management of file formats though normalization, there is an increasing interest towards this solution which could increase its adoption. P4 and iRODS, for different reasons, cannot be considered equivalent to the previous solutions, from the ForgetIT point of view. The former is still in a prototype status and is mainly focused on videos, although it integrates several useful technologies for AV digital preservation, while the latter was not conceived as an OAIS platform but could provide a valuable solution for the storage.

The selected platform will be integrated in the overall architecture and described in deliverable D8.2. The test infrastructure is currently in place and additional tests are required, based on actual data sets, which are in preparation. Additional limitations or advantages will be analysed when the other components will be available for integration.

The support of the ForgetIT data model will provide an additional criterion for the choice. The data model will be described in deliverable D8.2.

6 Middleware Solutions

The PoF Middleware will play a crucial role in the overall ForgetIT architecture, since it will provide the bridge between the active systems and the archive. Almost all components developed in the project by the technical WPs will be integrated in the Middleware, as part of one or more workflows. As a consequence, the middleware should be flexible and easily extensible and any unnecessary complexity should be avoided.

For the implementation of the PoF Middleware, we will adopt the following approach: for the early integration of the components we will use a lightweight solution, which should allow an easy integration of a few components in order to demonstrate a complete workflow starting from content creation and usage by active systems, submission to the middleware, processing through a limited number of components (e.g. the extractor and the contextualizer), ingest into the archive and final deposit into the cloud storage; then after content transformation in the cloud storage using storlets, the content will be brought back for active use (see also Section 4.2 for the details of the workflow). After this stage, we will try to integrate more complex and robust middleware solutions according to the requirements.

A possible candidate for the implementation of this lightweigth middleware component with minimal functionalities could be the workflow engine from P4 platform, which can be used to easily integrate command line tools and remote components available through REST interface. Another approach could be the use of a JMS component, such as the one included in Apache ServiceMix, to develop a Message-Oriented-Middleware (MOM), where different queues can be defined and a listener for each queue can invoke the appropriate component for processing. The results can be passed to the following queue through an XML message.

In the meanwhile different enterprise solutions will be evaluated, including full-fledged integration suites, integration platforms, application servers or simply Enterprise Service Bus (ESB) frameworks. Even if several commercial solutions implementing an enterprise-level middleware are available, the approach adopted in the project is to evaluate only free and Open Source solutions, supported by an active community of developers. There are currently several commercial ESB implementations on the market. However, many of these are built on top of an existing application server or messaging server, locking the implementation into a specific vendor.

Concerning the evaluation criteria, the middleware should provide, among other functionalities, the communication layer for all components, as well as the function to integrate components and to manage different business processes. Web interfaces must be provided for integration with other components of the architecture.

The main advantage of using an ESB component in the middleware is that it can act as a transit system for carrying data between applications which can be in the enterprise or spread across the web. The main benefit is that different applications can communicate with each other using a shared protocol. The ESB provides service creation and host-

ing, service mediation, message routing and data transformation, with exchange of data across varying formats and transport protocols.

Several candidate solutions have been proposed so far, based on previous experience and expertise from partners. Further investigation is in progress. According to the criteria above, the following solutions could be evaluated: Mule ESB[3], JBoss ESB[4], Apache ServiceMix[5], Apache UIMA[6], Cloud Foundry[7] and Taverna[8].

Mule ESB is a lightweight Java-based ESB and integration platform enabling quick and easy connection of applications and data exchange among them. As any ESB solutions, it aims at providing an easy integration of existing systems, regardless of the different technologies that the applications use. Examples of such technologies include JMS, Web Services, JDBC, HTTP, and more.

JBossESB is part of the JBoss Enterprise SOA Platform. The software is Enterprise Application Integration (EAI) or business integration software. The software is middleware used to connect systems together, especially non-interoperable systems, providing business process monitoring and management, connectors, transaction manager, security, application containers, messaging services, naming and directory service and others.

Apache ServiceMix is an Open Source integration container that unifies the features and functionality of several other components into a runtime platform, which can be used to build integration solutions. It provides a complete, enterprise ready ESB powered by OSGi Alliance, released under Apache License v2. Apache ServiceMix provides reliable messaging, routing and enterprise integration patterns, RESTful web services, a workflow engine based on BPEL, a JMS component and an orchestrator.

Taverna is an Open Source and domain-independent Workflow Management System. The Taverna suite is written in Java and includes the Taverna Engine (used for enacting workflows) that powers both the Taverna Workbench (the desktop client application) and the Taverna Server (which allows remote execution of workflows). Taverna is widely adopted to implement digital preservation workflows and has been used also in other projects related to digital preservation, such as SCAPE.

Two other solutions could be also considered, even if they implement different paradigms, because they provide useful features.

Apache UIMA is an unstructured information management application, a software system able to analyze large volumes of unstructured information in order to discover knowledge that is relevant to an end user. An example UIM application might ingest plain text and identify entities, such as persons, places, organizations; or relations. UIMA provides capabilities to wrap components as local or remote processors, to define data structures,

[3] http://www.mulesoft.org/
[4] http://www.jboss.org/jbossesb
[5] http://servicemix.apache.org/
[6] http://uima.apache.org/
[7] http://www.cloudfoundry.com/
[8] http://www.taverna.org.uk/

data flows and workflows to implement processing pipelines. It can scale to very large volumes by replicating processing pipelines over a cluster of networked nodes.

Cloud Foundry is an Open Source cloud computing platform as a service (PaaS) software, providing a choice of clouds, developer frameworks and application services, aiming at making faster and easier to build, test, deploy and scale applications. Cloud Foundry is developed by VMware and released under the terms of the Apache License 2.0.

The middleware solutions mentioned above will be evaluated and tested in the test environment and the results will be included in D5.2 [7].

7 Integration Plan

According to the DoW, three releases are expected for the ForgetIT framework, at M18 (D8.3), M27 (D8.4) and M36 (D8.6) respectively. In parallel two releases of the reference model will be available, at M15 (D8.2) and at M36 (D8.5) respectively.

7.1 Plan for the first ForgetIT release

For the first release of the integrated framework we will adopt a two-step approach: we will start with the integration of a limited number of components in order to demonstrate the two identified simple integrated workflows (see Section 4.2) involving the four main blocks of the architecture . This early integration will demonstrate that a complete workflow can be executed, starting from active content use, through managed forgetting, contextualization, archiving and storage and pushing of the content back to active use after transformation. With this infrastructure in place, we will be able to test the main interfaces provided by each system and the communication among the different layers. From this we expect the elicitation of further requirements with respect to the interaction between components. We will also evaluate the packaging model used for archiving and the final deposit on the cloud storage. After this phase will be completed, we will add more components to the middleware, to add additional functionalities. According to the plan for the development of the components, for almost all of them a first version will be ready by the time of the first release. They will be improved and extended during the project lifetime, also taking into account evaluation results from the applications (pilots in month 23). Concerning the middleware implementation, the most appropriate solution will be adopted, focusing on actual integration of the components and trying to avoid unnecessary complication.

In a nutshell, for the first release the project will deliver a preliminary version of the framework with the main architecture blocks already integrated and able to communicate. This will be used to demonstrate two initial integrated priority workflows for both the personal and organizational scenario.

7.2 Preliminary plan for the other releases

A detailed integration plan for the other releases is not possible at this time. We can expect that after the first platform release, the evaluation from the users will provide additional feedbacks which will help in improving the overall system. The further development of the components will also provide additional features to be integrated. The plan for the other releases will be defined after completing the first release following the lines described in the DoW.

8 Test Environment

The activities related to development and integration of architecture components will be supported by a test environment which is already available to all partners. The test environment depicted in Figure 7, hosted by EURIX, will be available from outside only to project partners using a VPN connection to grant access to a ForgetIT dedicated private network. This will guarantee the appropriate level of confidentiality for all development activities under the consortium agreement.

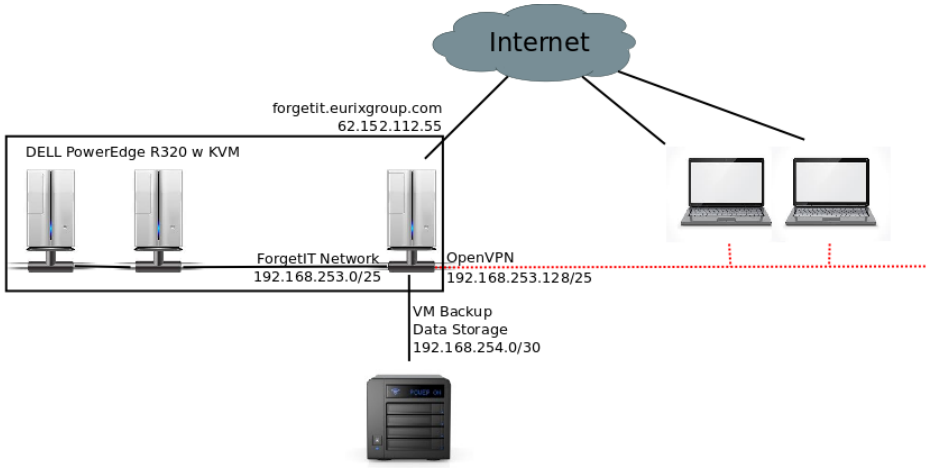

Figure 7: Configuration of the test environment

The test environment includes a DELL PowerEdge R320 server, equipped with Linux (Ubuntu 12.04 LTS 64-bit), and a 8 TB NAS for data storage, connected via dedicated Gb Ethernet connection.

The testbed server provides a virtualization environment based on Kernel-based Virtual Machine (KVM) [44]. KVM is a full virtualization solution for Linux on x86 hardware containing virtualization extensions (Intel VT or AMD-V) and consists of a loadable kernel module providing the core virtualization infrastructure and a processor specific module. Using KVM, one can run multiple virtual machines running unmodified Linux or Windows images. Each virtual machine has private virtualized hardware: a network card, disk, graphics adapter, etc. The kernel component of KVM is included in the main Linux kernel since version 2.6.20.

Each system which is part of the overall ForgetIT architecture will be provided as a new virtual machine for KVM (qcow2 format) and will be available on the dedicated network accessible via VPN. Each system or component to be tested and integrated in the ForgetIT platform will be provided by each partner in a virtual machine and will be uploaded using an FTP service (secured by VPN) for being installed.

A ForgetIT dedicated network has been created, the IP addresses of the different services available will be shared among all partners. An FTP area is also available for uploading

and sharing huge files for the integration and test (e.g. virtual machines, installers, configuration files, test samples, ...). The initial activities of development and test for each component will be performed by each partner at their own institution. For example the cloud storage services based on OpenStack Swift will be initially developed and tested at IBM premises and when a new release is ready for testing by other partners will be deployed to the test environment. For specific purposes or events, the partners could decide to deliver part of or the whole system to other servers and infrastructures, for example in case of specific testbed events which require datasets or other resources which cannot be supported or managed in the test environment.

9 Conclusions and future work

In this deliverable we have described the PoF architecture, the main systems and components developed and integrated in the project as well as some candidate solutions for implementing the main building blocks of the overall architecture. A preliminary plan for integrating and testing the different results provided by the technical work packages has been included and the setup of the test environment which is being used for development and testing has been also described.

In the next WP8 deliverables, the reference model and the framework implementation based on the present architecture will be reported. In particular, deliverable D8.3 (expected at M18) will describe the first release of the framework, with integrated components, supported workflows and content types as well as the adopted solutions for the archive and the middleware. The first release of the platform will be tested and the collected feedbacks will be included in the second release, expected at M27 (reported in deliverable D8.4).

References

[1] ForgetIT. D3.1 - Report on Foundations of Managed Forgetting. http://www.forgetit-project.eu/fileadmin/fm-dam/deliverables/ForgetIT_WP3_D3.1.pdf, August 2013.

[2] ForgetIT. D4.1 - Information Analysis, Consolidation and Concentration for Preservation - State of the Art and Approach. http://www.forgetit-project.eu/fileadmin/fm-dam/deliverables/ForgetIT_WP4_D4.1.pdf, July 2013.

[3] ForgetIT. D5.1 - Foundations of Synergetic Preservation. http://www.forgetit-project.eu/fileadmin/fm-dam/deliverables/ForgetIT_WP5_D5.1.pdf, July 2013.

[4] ForgetIT. D6.1 - State of the Art and Approach for Contextualization. http://www.forgetit-project.eu/fileadmin/fm-dam/deliverables/ForgetIT_WP6_D6.1.pdf, July 2013.

[5] ForgetIT. D7.1 - Foundations of Computational Storage Services. http://www.forgetit-project.eu/fileadmin/fm-dam/deliverables/ForgetIT_WP7_D7.1.pdf, July 2013.

[6] ForgetIT. D9.1 - Application Use Cases & Requirements Document. http://www.forgetit-project.eu/fileadmin/fm-dam/deliverables/ForgetIT_WP9_WP10_D9.1.pdf, August 2013.

[7] ForgetIT. D5.2 - Workflow Model and Prototype for Transition between Active System and AIS, Expected on February 2014.

[8] OASIS. Content Management Interoperability Services. https://www.oasis-open.org/committees/cmis. Retrieved on 31 October 2013.

[9] CCSDS. Reference Model for an Open Archival Information System. http://public.ccsds.org/publications/archive/650x0m2.pdf. Retrieved on 31 October 2013.

[10] TYPO3. Source code repository. http://typo3.org/about/typo3-the-cms. Retrieved on 31 October 2013.

[11] Library of Congress. Metadata Encoding and Transmission Standard. http://www.loc.gov/standards/mets/. Retrieved on 31 October 2013.

[12] ENSURE Project. http://ensure-fp7-plone.fe.up.pt/site/. Retrieved on 31 October 2013.

[13] ENSURE Project. PDS Interface Specification. http://ensure-fp7-plone.fe.up.pt/site/deliverables/pds-cloud-external-interface-specification/at_download/file. Retrieved on 31 October 2013.

[14] Technology Readiness Level. http://en.wikipedia.org/wiki/Technology_readiness_level. Retrieved on 31 October 2013.

[15] United States Department of Defense. Technology readiness assessment (tra) guidance. http://www.acq.osd.mil/chieftechnologist/publications/docs/TRA2011.pdf, April 2011. Retrieved on 31 October 2013.

[16] DSpace SourceForge Repository. https://sourceforge.net/projects/dspace/files. Retrieved on 31 October 2013.

[17] DSpace GitHub Repository. https://github.com/DSpace/DSpace. Retrieved on 31 October 2013.

[18] DSpace Documentation. https://wiki.duraspace.org/display/DSDOC3x/DSpace+3.x+Documentation. Retrieved on 31 October 2013.

[19] DuraSpace. http://www.duraspace.org. Retrieved on 31 October 2013.

[20] DSpace Data Model. https://wiki.duraspace.org/display/DSDOC3x/Functional+Overview#FunctionalOverview-DataModel. Retrieved on 31 October 2013.

[21] Keep Solutions. http://www.keep.pt. Retrieved on 31 October 2013.

[22] RODA GitHub Repository. https://github.com/keeps/roda. Retrieved on 31 October 2013.

[23] RODA Documentation. https://github.com/keeps/roda/wiki/Developer-guide. Retrieved on 31 October 2013.

[24] Artefactual Systems. http://www.artefactual.com. Retrieved on 31 October 2013.

[25] Archivematica GitHub Repository. https://github.com/artefactual/archivematica. Retrieved on 31 October 2013.

[26] Archivematica Documentation. https://www.archivematica.org/wiki/Documentation. Retrieved on 31 October 2013.

[27] elasticsearch. Open Source Distributed Real Time Search & Analytics. http://www.elasticsearch.org. Retrieved on 31 October 2013.

[28] Fedora GitHub Repository. https://github.com/fcrepo. Retrieved on 31 October 2013.

[29] Fedora Documentation. https://wiki.duraspace.org/display/FEDORA37/Fedora+3.7+Documentation. Retrieved on 31 October 2013.

[30] Fedora Digital Object Model. https://wiki.duraspace.org/display/FEDORA37/Fedora+Digital+Object+Model. Retrieved on 31 October 2013.

[31] PrestoPRIME Project. http://www.prestoprime.eu. Retrieved on 31 October 2013.

[32] P4 Repository. https://github.com/prestoprime/p4. Retrieved on 31 October 2013.

[33] P4 - PrestoPRIME Preservation Platform. http://prestoprime.eurixgroup.com/p4. Retrieved on 31 October 2013.

[34] P4 Documentation. https://www.prestocentre.org/library/tools/p4. Retrieved on 31 October 2013.

[35] iRODS Documentation. https://www.irods.org/index.php/Documentation. Retrieved on 31 October 2013.

[36] DELOS Project. http://www.dpc.delos.info. Retrieved on 31 October 2013.

[37] CASPAR Project. http://www.casparpreserves.eu. Retrieved on 31 October 2013.

[38] PLANETS Project. http://www.planets-project.eu. Retrieved on 31 October 2013.

[39] Shaman Project. http://shaman-ip.eu/. Retrieved on 31 October 2013.

[40] SCAPE Project. http://www.scape-project.eu. Retrieved on 31 October 2013.

[41] TIMBUS Project. http://timbusproject.net/. Retrieved on 31 October 2013.

[42] Presto4U Project. https://www.prestocentre.org/4u. Retrieved on 31 October 2013.

[43] Digital Preservation Coalition. http://www.dpconline.org/. Retrieved on 31 October 2013.

[44] Kernel-based Virtual Machine. http://www.linux-kvm.org. Retrieved on 31 October 2013.