www.forgetit-project.eu

ForgetIT

Concise Preservation by Combining Managed Forgetting and Contextualized Remembering

Grant Agreement No. 600826

Deliverable D8.3

Work-package	WP8: The Preserve-or-Forget Reference Model and Framework
Deliverable	D8.3: The Preserve-or-Forget Framework – First Release
Deliverable Leader	Francesco Gallo (EURIX)
Quality Assessor	Alexander Damhuis (dkd)
Estimation of PM spent	20
Dissemination level	PU
Delivery date in Annex I	M18 (July 2014)
Actual delivery date	29 August 2014
Revisions	8
Status	Final Draft
Keywords	PoF Framework, first prototype, integrated components

Revision History

Version	Major changes	Authors
v0.8	Implemented QA comments. Added section about evaluation of deliverable using WP8 success/progress indicators. Final reading.	L3S, dkd, EURIX
v0.7	Added description of Collector, Archiver, ID Manager and Scheduler. Added class diagrams.	EURIX
v0.6	Added description of Enterprise Integration Patterns and message routing, data and workflow management, OAIS concepts.	EURIX
v0.5	Added description for Digital Repository and Cloud Storage, first prototype implementation, software development, deployment and testing.	IBM, EURIX
v0.4	Described PIMO, TYPO3 and Contextualizer.	DFKI, dkd, USFD
v0.3	Described PoF Middleware, Extractor and Forgettor.	L3S, CERTH, EURIX
v0.2	Completed Executive Summary, overview of architecture, Message Oriented Middleware concepts and technologies.	EURIX
v0.1	Created ToC and defined main Sections. Assigned contributions.	EURIX

List of Authors

Partner Acronym	Authors
LUH	Claudia Niederee, Nattiya Kanhabua
IBM	Doron Chen
DFKI	Heiko Maus
CERTH	Vasilis Solachidis
dkd	Alexander Damhuis
USFD	Mark A. Greenwood
EURIX	Francesco Gallo, Jacopo Pellegrino

Table of Contents

Executive summary

The present deliverable consists of two parts: (1) the implementation of the first release of the Preserve-or-Forget (PoF) Framework and (2) the description of this framework.

The first release of the Preserve-or-Forget (PoF) Framework is based on the architecture and integration plan defined in D8.1 [ForgetIT 2013d], integrates the components developed in the technical WPs and provides a foundation for application pilot development in WP9 and WP10.

The implemented APIs, used to integrate the different architecture components, were defined in D8.1. We leverage the CMIS specification to retrieve content from Active Systems, while the PoF Middleware exposes REST APIs which are used to trigger the preservation workflows and other processes in the PoF Framework.

The PoF Middleware has been implemented as a Message Oriented Middleware (MOM). We provide a short discussion about the chosen approach and some basic information about technologies used to implement the messaging system and the middleware bus.

For the components identified in D8.1, either providing common tasks or implementing core ForgetIT functionality, the mechanism for their integration is provided. For each component we describe the status, the main APIs, the input and output formats and the integration details, while we refer to either D8.1 or to the relevant deliverables in the corresponding WP to describe the purpose and the functionalities of each component.

We then briefly discuss the workflow management in the middleware, describing the exchange of messages among components and the main end-to-end reference scenario for synergetic preservation and managed forgetting which has been implemented in the first prototype release. Further improvements to the workflow management are also outlined, for example those related to the use of Enterprise Integration Patterns (EIP) or to the integration of additional ESB components on top of the existing solution.

Concerning the Preservation System, we describe the two main components, the Digital Repository and the Cloud Storage System. Both systems implement an OAIS compliant solution for the preservation of ForgetIT content. The APIs exposed by the Preservation System are discussed and the implementation using DSpace and PDS is described. For the Digital Repository and the Cloud Storage we describe how they implement the OAIS functional entities and the internal data models used by both systems. Finally we describe how Storlets are involved in the current workflow. We also describe how the information packages are created and provide some examples.

Finally we provide additional information about the software development process and the collaborative tools, as well as preliminary considerations about the license of the overall PoF Framework. The software documentation for the PoF Middleware and the Preservation System APIs are available on the project web site.

1 Introduction

The main topic of this document is the description of the first prototype implementation of the PoF Framework. This prototype is the result of the activities performed during the first year within Task 8.3 - Development of the PoF Framework.

The first PoF Framework prototype provides an integration framework for all available components, is based on the ForgetIT architecture and is used to validate basic workflows for managed forgetting and synergetic preservation. This deliverable consists of the description of the prototype in this document as well as of the prototype itself, which is running on the servers at EURIX.

According to the project proposal, the prototype implementation is based on open and widely adopted technologies, using an integration approach which guarantees flexibility and loose coupling among all the components.

The development has been performed in a collaborative way, sharing a code repository and tracking open issues, discussed in periodic meetings by all interested partners.

The architecture of the PoF Framework is described elsewhere (see deliverable D8.1 [ForgetIT 2013d]), in this document we just summarize the relevant information for the sake of clarity. The components, the interfaces and protocols used to design the overall framework have been identified during the first year. With respect to the original project plan, the assessment of candidate technologies for the PoF Middleware and the Digital Repository was anticipated to the first year, in order to provide as early as possible a complete end-to-end solution with all architecture layers integrated together. During the second and third year of the project, missing components will be integrated and the available ones will be improved. The evaluation of the framework performances based on a set of benchmark scenarios will also be part of the future development activities.

The PoF Framework will also be compliant to the PoF Reference Model, which is still in preparation at the time of writing. The aim of this model is to go beyond the constraints of OAIS [CCSDS 2012], which is commonly adopted as the reference specification for archival systems in digital preservation field but is lacking appropriate definition of the core ForgetIT principles (synergetic preservation, managed forgetting and contextualized remembering). A discussion of the OAIS role in the ForgetIT architecture is provided in deliverable D8.1 [ForgetIT 2013d], while the PoF Reference Model is described in deliverables D8.2 [ForgetIT 2014f] and D8.5 [ForgetIT 2015b].

The implementation of the first prototype leverages the analysis of workflow models for synergetic preservation, reported in deliverable D5.2 [ForgetIT 2014c]. The definition of information packages created in the PoF Middleware and imported in the Preservation System is based on the results provided by WP5. Several components provided by technical WPs and integrated in the prototype have already been described in detail in corresponding deliverables, namely D3.2 [ForgetIT 2014a], D4.2 [ForgetIT 2014b], D5.2 [ForgetIT 2014c] and D6.2 [ForgetIT 2014d] for the PoF Middleware components, D7.1 [ForgetIT 2013c] and D7.2 [ForgetIT 2014e] for the Cloud Storage, D8.1 [ForgetIT

2013d] for the Digital Repository and finally D9.2 [ForgetIT 2014g] and D10.1 [ForgetIT 2013a] for the Active Systems.

In the following Sections, the component description will focus on those aspects which are relevant for integration, such as APIs and I/O formats and protocols, while for component implementation details please refer to the relevant deliverables mentioned above.

The document is organized as in the following: we start with an assessment of the results with respect to the success indicators, which have been defined in the ForgetIT Description of Work (Section 2); a summary of the relevant information concerning the PoF Framework architecture is reported in Section 3, while the relevant architecture blocks are described in Section 4 and 5 (PoF Middleware), Section 6 (Active Systems) and Section 7 (Preservation System); the prototype implementation, including software development, documentation and license, is reported in Section 8; in Section 9 we describe the future activities towards the second framework release; finally we added an appendix to describe a simple demo with application screenshots and two appendices with installation and configuration instructions concerning DSpace (used to implement the Digital Repository) and Apache ActiveMQ (used for the messaging system in the PoF Middleware).

Target audience for this deliverable

This deliverable targets a technically oriented readership, which is interested in the technical aspects of the implementation of the PoF Framework, plans to adopt the framework or wants to use it as a blueprint for a similar project.

2 Assessment of WP8 indicators

The expected WP8 outcomes, reported in the project proposal, are:

- the Preserve-or-Forget (PoF) Reference Model
- the PoF Framework

The results described here for the first framework prototype refer to outcome 2, for which the following success/progress indicators have been identified in the project proposal:

1. *availability of interfaces and protocols exposed/published by software components to be integrated and delivered by technical work packages*,
2. *adequateness and effectiveness of the defined integration approach and strategy for the occurring integration tasks*,
3. *availability of infrastructure facilities for managing the development of the software framework (e.g. versioning system, software repository)*.

The first prototype implementation has been shown at the first annual review. We consider the results presented here in agreement with all indicators above and the objectives associated to all indicators as adequately satisfied, as described below.

Indicator 1: APIs and protocols

The APIs and protocols of the components to be integrated have been defined, the first prototype integrates the components according to the integration plan in D8.1. The APIs published by the PoF Middleware and the Preservation System (Digital Repository and Cloud Storage Service) are based on REST architectural style, hence different HTTP verbs are used to get and send data. The REST APIs, described in the software documentation, have been implemented using Java reference software (Jersey), for maximum compatibility with all external systems. Concerning the protocols, CMIS is used to retrieve resources and metadata from Active Systems. CMIS is a open standard protocol aimed to support interoperability and is widely adopted and supported. Standard formats have been used for content packaging (XML-based formats such as METS, Dublin Core, PREMIS) and for communication with web services (XML or JSON).

Indicator 2: integration approach

The integration approach leverages the best practices in Enterprise Application Integration (EAI), adopting well established concepts such as the Enterprise Service Bus (ESB) for the communication layer and Enterprise Integration Patterns (EIP) as industry level standard for complex integration patterns. Benefits of such approach are discussed in the document. Concerning the actual implementation, a message oriented approach has been chosen for the integration framework. The PoF Middleware is implemented as a Message Oriented Middleware (MOM). The adopted approach has been already validated integrating heterogeneous components delivered as binary executables, libraries or external services, to implement the priority workflows described in D8.1. The current prototype includes examples for each category of integrated components.

Indicator 3: development and test infrastructure

The testbed environment has been setup and is accessible to all partners. We made extensive use of virtualization for testing components developed by each partner. The software developed is managed using a versioning system (Subversion), available on the project workspace and accessible to all partners. An issue tracking system (Trac) is used to keep track of all open issues and for ticketing, as well as to define milestones for the development (software releases, deadlines, etc.) and to share information about exceptions and errors. Each ticket is assigned to the appropriate partner. Progress for each milestone and deadline can be monitored, taking into account open tickets. The approach adopted for software development is based on Agile methodology, using UML for sharing ideas and to describe software components, from preliminary sketches to complex modules. Only a minimal amount of documents is created during the development phase, leveraging the software repository and the issue tracking system. SCRUM is used as favorite approach, with periodic conference calls among all partners involved in the integration and to plan the next steps, inspired by Sprint Planning meetings. The tracking system provides the Dashboard, while dedicated sessions during plenary project meeting have been used to define component requirements and functionalities (Product Backlog).

3 Preserve-or-Forget Framework

In this Section we quickly remind the main features of the PoF Framework architecture, already discussed in deliverable D8.1 [ForgetIT 2013d]. The UML component diagram of the overall architecture is depicted in Figure 1, where the three layers are shown: Active Systems, PoF Middleware and Preservation System.

Concerning the active components, the Semantic Desktop component is related to the personal preservation and contains the PIMO. It provides a user interface for both desktop and mobile users. The TYPO3 CMS is related to the organizational preservation. It provides a desktop user interface too and contains the CMIS repository within the adapter. Both the Active Systems have an adapter in order to interact with the middleware REST APIs. Contents from the Active Systems are retrieved through CMIS interface. Additional information about the Active Systems can be found in Section 6.

The active components exchange information with the PoF Middleware. The middleware component enables the decoupling between the Active Systems and the Preservation System. Part of the components within the PoF Middleware, such as the the Scheduler or the ID Manager, are related to workflows and data management. Other components extract from data those features necessary for SIP creation. They are, for instance, the Navigator, the Extractor, the Forgettor and the Collector/Archiver. In particular, the Collector/Archiver interacts with the Preservation System through the REST interfaces enabling both SIP ingest and access of AIPs stored in the Digital Repository. Further information about both the concepts of MOM and the PoF Middleware is available in Section 4.

The ingest and access processes are managed, on the Preservation System side, by the Package Importer and Exporter components. The functionalities of the Digital Repository are provided by the Repository Manager (with an internal Data Manager) and by the Preservation Manager. Storage functionalities are provided by the Cloud Storage Service which includes the Preservation DataStores (PDS), made up of the Preservation Engine and the OpenStack Swift component, which contains the Storlet Engine. The Preservation System is described in Section 7.

For the implementation of the first framework prototype, the integration of the components followed the integration plan reported in deliverable D8.1 [ForgetIT 2013d], the list of components included in the first release is shown in Table 15 of D8.1.

Two priority workflows have been implemented, for basic synergetic preservation and for managed forgetting support, according to the diagrams reported in Figure 5 and Figure 6 of deliverable D8.1. Those two priority workflows involve all layers in the architecture, from preservation requests for content generated in the user application to permanent storage in the Preservation System, passing through a number of steps in each workflow where different PoF Middleware components are activated for e.g. extracting features or generating the context.

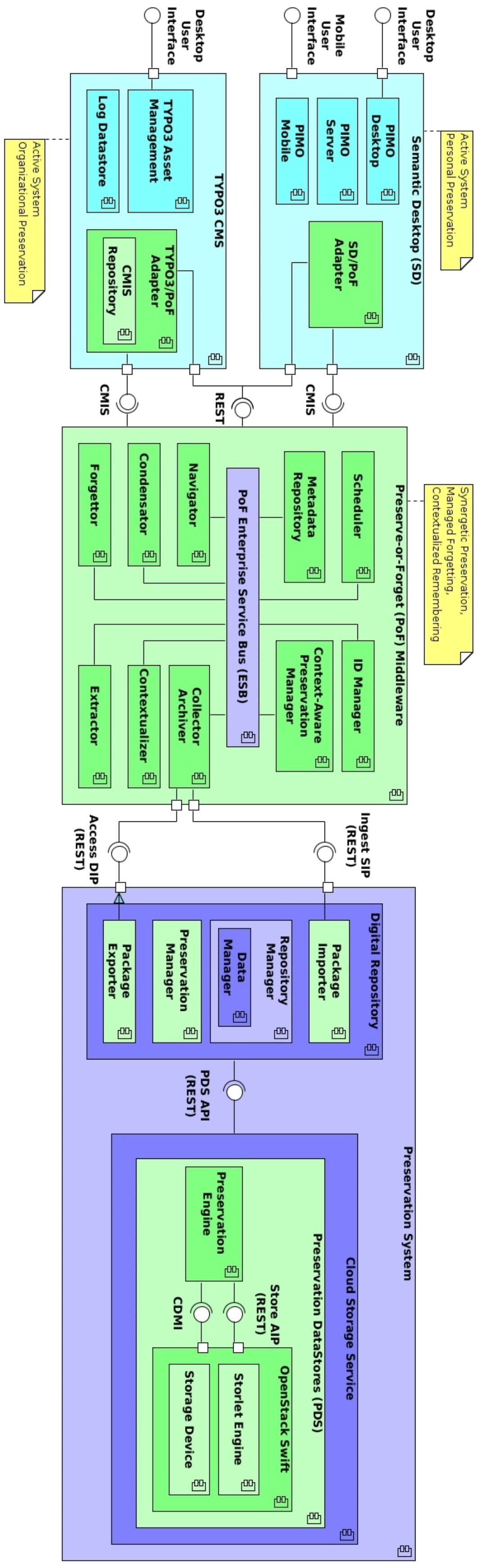


Figure 1: PoF Framework Component Diagram

4 PoF Middleware Implementation

In the following, the message oriented approach, which is used for implementing the PoF Middleware, will be discussed.

4.1 Message Oriented Middleware

The concept of MOM has already been introduced in deliverable D5.2 [ForgetIT 2014c] and suggested as possible approach in deliverable D8.1 [ForgetIT 2013d]. One of the main advantages provided by MOM is the loose coupling among integrated components.

A MOM lies between the applications acting as a message mediator between them [Snyder u. a. 2011] by means of a communication channel that carries self-contained units of information which are the messages [Chappell 2004]. It is possible to say that the MOM mediates events and messages amongst distributed systems providing the required degree of decoupling. Figure 2 provides a view of this kind of architecture.

According to [Snyder u. a. 2011], a MOM could be best described as a category of software for communication in an loosely-coupled, reliable, scalable and secure manner amongst distributed applications or systems. In an enterprise environment, several applications may interact and exchange data. In case the information exchange takes place directly among the distributed applications a connection between them exists which may not be desirable: in such a scenario the failure of one application could cause the failure of all the others. This connection is usually referred as *coupling* and indicates the interdependence of two or more applications or systems [Snyder u. a. 2011].

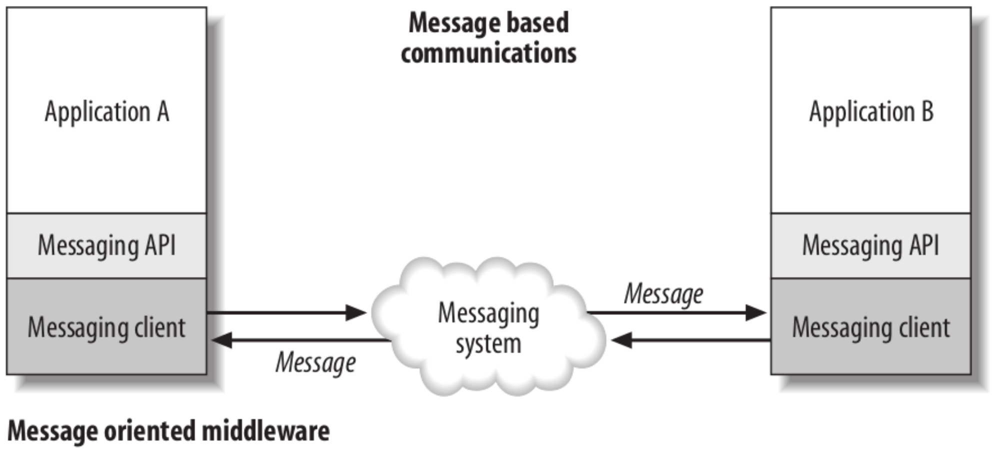


Figure 2: Message based communication [Chappell 2004].

Figure 2 also shows how senders know nothing about receivers and receivers know nothing about senders. This is commonly known as asynchronous messaging [Chappell 2004, Snyder u. a. 2011].

A *Producer* is allowed to send a message that will be delivered either to one or many *Consumers*. In the former case the message is sent through a specific *queue* and the messaging model is named *Point-to-Point*, in the latter the message is published on a *topic* and the model is named *Publish-and-Subscribe* [Chappell 2004]. Figure 3 depicts the two messaging models.

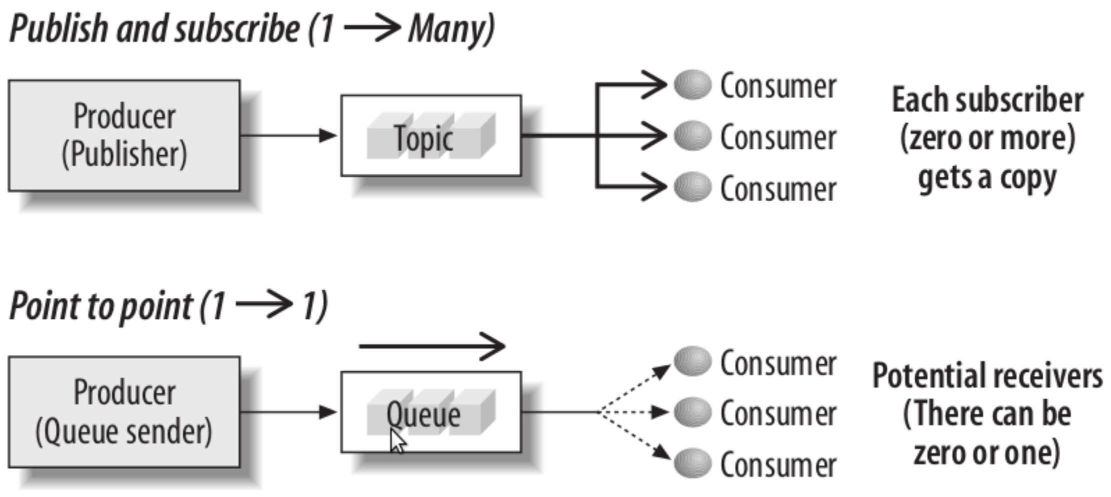


Figure 3: Point-to-Point and Publish-and-Subscribe models [Chappell 2004].

The MOM also has the responsibility to ensure that the messages reach their intended destination and that they are not lost in case of network failure [Chappell 2004, Snyder u. a. 2011], therefore the messages have to be stored into a persistent memory and accessed when requested from the *Consumer*. This feature is referred to as *message persistence*. Figure 4 depicts an example of message exchange where the *Consumer* looses the connection to the MOM but the message does not get lost.

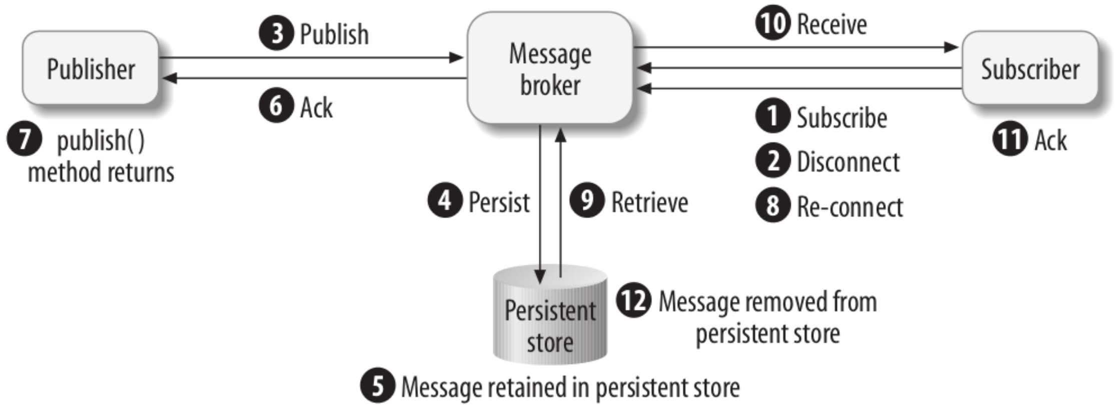


Figure 4: Example of message persistence [Chappell 2004].

The MOM approach is a suitable solution for the management and the integration of the various components in the project, where several heterogeneous components are integrated in a middleware and asynchronous communication is a requirement. If the

MOM provides a reliable and flexible communication infrastructure, we need to organize the data flow and task execution with messages in order to implement complex workflows.

For the implementation of the different workflows, we will make use of Enterprise Integration Patterns (EIP), defined in the fundamental book by G. Hohpe [Hohpe und Woolf 2003]. The EIP approach has been extensively adopted to design asynchronous messaging architectures used to build integration solutions and is used in several enterprise-class applications. The book describes 65 design patterns for the use of Enterprise Application Integration (EAI) and MOM in the form of a pattern language. They are accepted solutions to recurring problems within a given context. Patterns are abstract enough to apply to most integration technologies, but specific enough to provide hands-on guidance to designers and architects. Patterns also provide a vocabulary for developers to efficiently describe their solution. Patterns are not 'invented'; they are harvested from repeated use in practice. A coherent collection of relevant patterns that form an integration pattern language is available on the EIP web site[1].

4.2 Technologies for PoF Middleware

The PoF Middleware has been implemented using Apache ActiveMQ[2]. ActiveMQ is an open source, JMS 1.1 compliant MOM from the Apache Software Foundation that provides high-availability, performance, scalability, reliability and security for enterprise messaging [Snyder u. a. 2011]. It also provides all the MOM functionalities allowing the user to implement and customize specific message producers and consumers that exchange information through queues and topics. ActiveMQ is commonly adopted in enterprise scenarios when an asynchronous message bus is needed (see for example [Henjes u. a. 2007, DAI und ZHU 2010] and other references available in the literature).

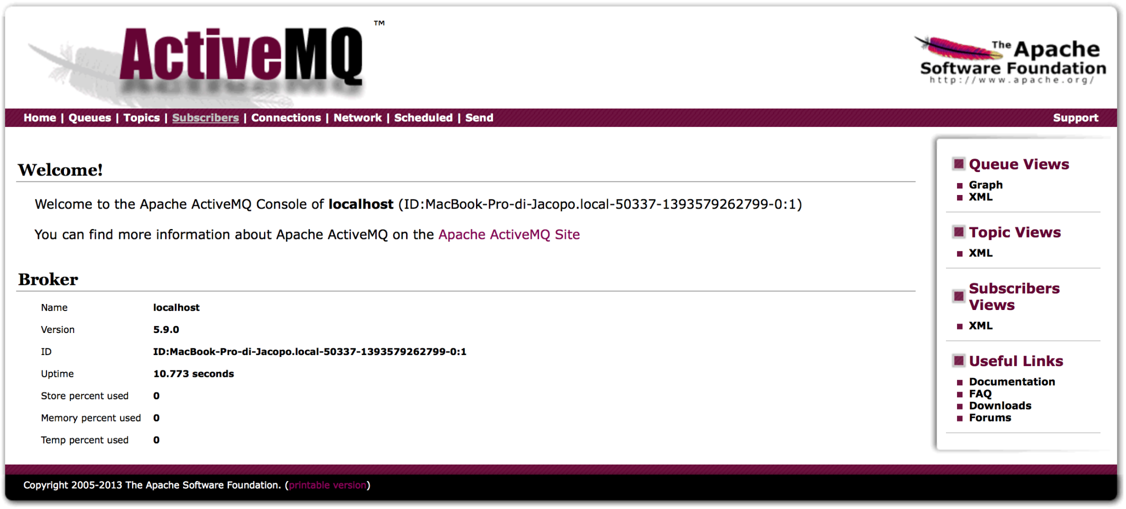
In our configuration ActiveMQ runs as a web application inside Tomcat (see Appendix C). The ActiveMQ webapp also provides a GUI for the user to monitor the status of queues and topics and the message exchange through them, as shown in Figure 5.

On top of the message broker implemented by ActiveMQ, a rule-based routing and mediation engine will be added in the next release of the PoF Middleware, in order to implement the middleware workflows using one of the EIPs. The rule engine is provided by Apache Camel[3] and is briefly described in Appendix C.

4.3 PoF Middleware APIs

As will be described in Section 8, the package `eu.forgetit.middleware` of the PoF Middleware Java project contain the main classes for the implementation of the PoF Mid-

[1]Enterprise Integration Patterns - `http://www.eaipatterns.com/`
[2]Apache ActiveMQ - `http://activemq.apache.org`
[3]Apache Camel - `http://camel.apache.org`


ActiveMQ™
The Apache Software Foundation http://www.apache.org/
Home | Queues | Topics | Subscribers | Connections | Network | Scheduled | Send
Support
Queue Views
Graph
XML
Topic Views
XML
Subscribers Views
XML
Useful Links
Documentation
FAQ
Downloads
Forums

Welcome!

Welcome to the Apache ActiveMQ Console of **localhost** (ID:MacBook-Pro-di-Jacopo.local-50337-1393579262799-0:1)

You can find more information about Apache ActiveMQ on the Apache ActiveMQ Site

Broker

Name	localhost
Version	5.9.0
ID	ID:MacBook-Pro-di-Jacopo.local-50337-1393579262799-0:1
Uptime	10.773 seconds
Store percent used	0
Memory percent used	0
Temp percent used	0

Copyright 2005-2013 The Apache Software Foundation. (printable version)

Figure 5: Screenshot of the ActiveMQ web GUI.

dleware. In particular, the class `ConnectionFactory` contains all the methods necessary to set and monitor the JMS Connection to the broker. The excerpt below lists all methods of the class to manage the connection.

```
public class ConnectionFactory {

        private static Logger logger = LoggerFactory.getLogger
        (ConnectionFactory.class);

        private static ActiveMQConnectionFactory connection
        Factory = null;

        private static Connection connection = null;
        private static boolean connected = false;

        private static List<MessageConsumer> consumers = null;

        public static Session getSession() throws JMSException{...

        public static void addConsumer(MessageConsumer
        consumer){...

        /*
         * Check TCP socket connetion to broker using NIO socket
         * channel
         */
        private static void checkConnection(String host, int
        port) {...

        public static void closeConnection() {...
        }

}
```

The `QueueManager` manages the operations related to the topics or the queues through which the messages are exchanged. Here below the methods to create the queues and instantiate the processors and to send and consume the messages are listed.

```
public class QueueManager {

        private static Logger logger = LoggerFactory.getLogger
        (QueueManager.class);
        private static QueueManager queueManager = null;
        private static Configuration configuration = null;
        private static Producer producer = null;
        private static List<Thread> threadList = null;

        public static synchronized QueueManager getInstance(){...

        private QueueManager(){...

        public void sendMessage(String message, String
        queueName) throws JMSException{...

        public void sendMessage(Message message, String
        queueName){...

        public MessageConsumer getConsumer(String queueName)
        throws JMSException{...

        private synchronized void init(){...

        private void thread(Runnable runnable, String name,
         boolean daemon){...

        public void stopBrokerThreads(){...

}
```

The class `Producer` and the abstract class `Processor` implement the `Runnable` interface (used for multi-threading) for respectively generate and process a message. It is also possible to set a specific queue as either input or output for the messages.

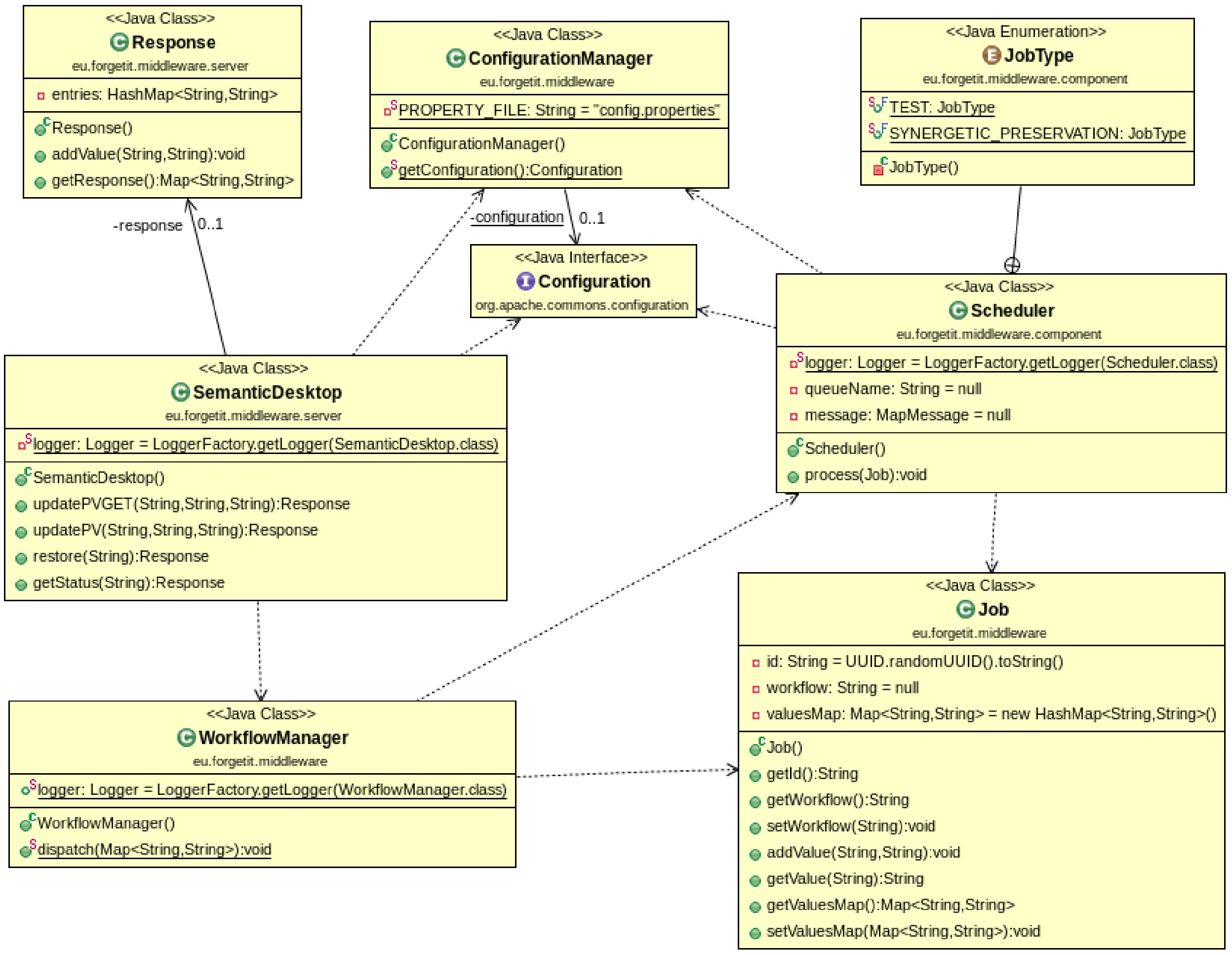
```
public class Producer implements Runnable{

        private static final Logger logger = LoggerFactory.
        getLogger(Producer.class);
        private Session session = null;
        private Message message = null;
        private String queue = null;

        public Producer(Session session, Message message,
        String queue)

        public void run() {...

}
```

```
public abstract class Processor implements Runnable,
MessageListener {

        private String inputQueue = null;
        private String outputQueue = null;

        public Processor(String inputQueue,String
        outputQueue){...

        // this method must be implemented by actual processor,
        // executing business logic for that component
        public abstract void processMessage(TextMessage message)
        throws JMSException ;
        public abstract void processMessage(MapMessage message)
        throws JMSException;

        public void run() {...

        @Override
        public void onMessage(Message message) {...

        public String getInputQueue() {...

        public String getOutputQueue() {...

        public MapMessage getMutableMapMessage(MapMessage
        message){...

}
```

A `MessageLogger` can also be instantiated in order to log all the operations performed by the producers and processors in a given JMS Session. The methods of the class are shown here below.

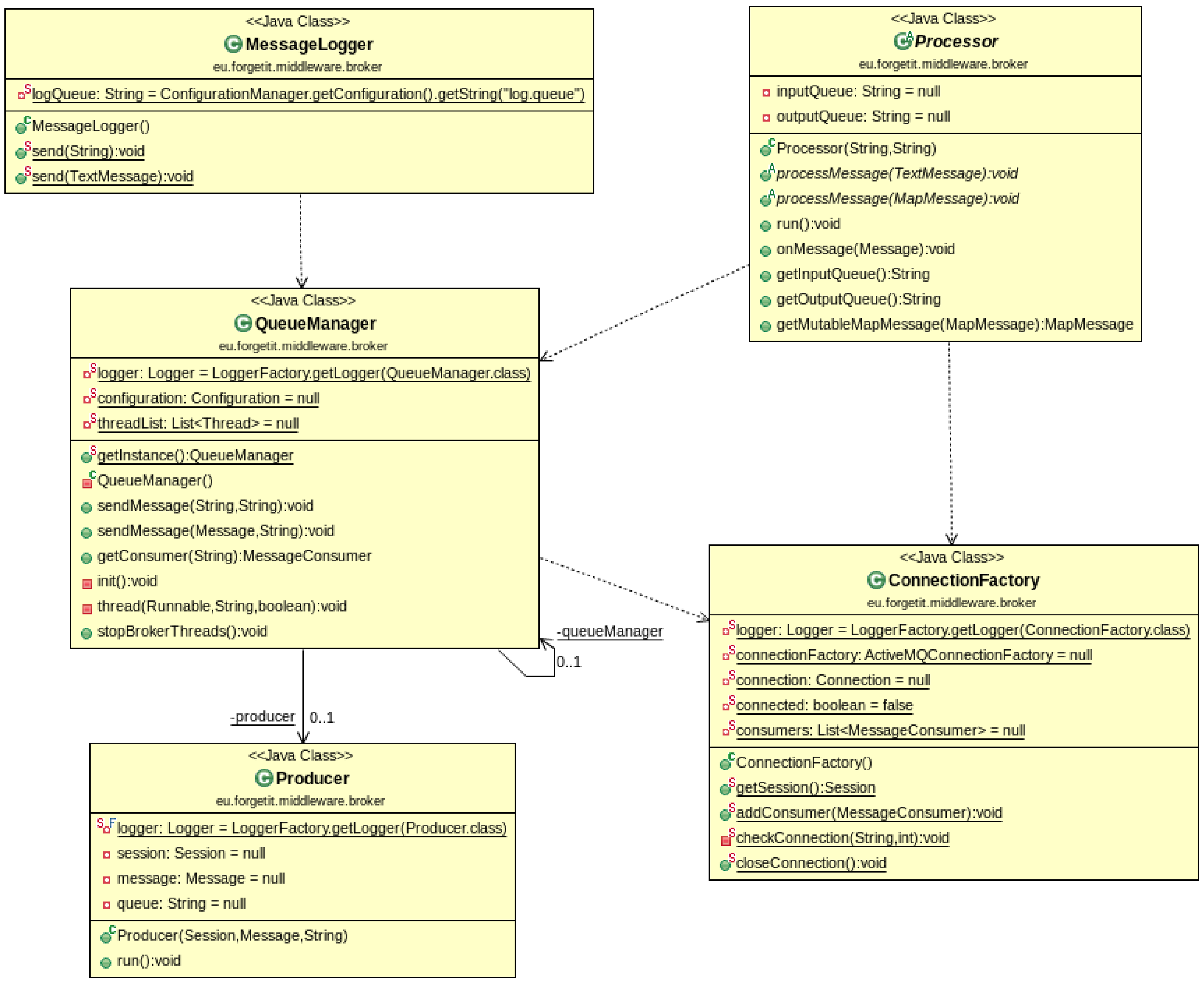
```
public class MessageLogger {

        private static String logQueue = ConfigurationManager.
        getConfiguration().getString(”log.queue”);

        public static void send(String message){...

        public static void send(TextMessage message){...

}
```

The main classes of the PoF Middleware are shown in Figure 6, Figure 7 and Figure 8. Additional information can be found in the software documentation, see Section 8.2.



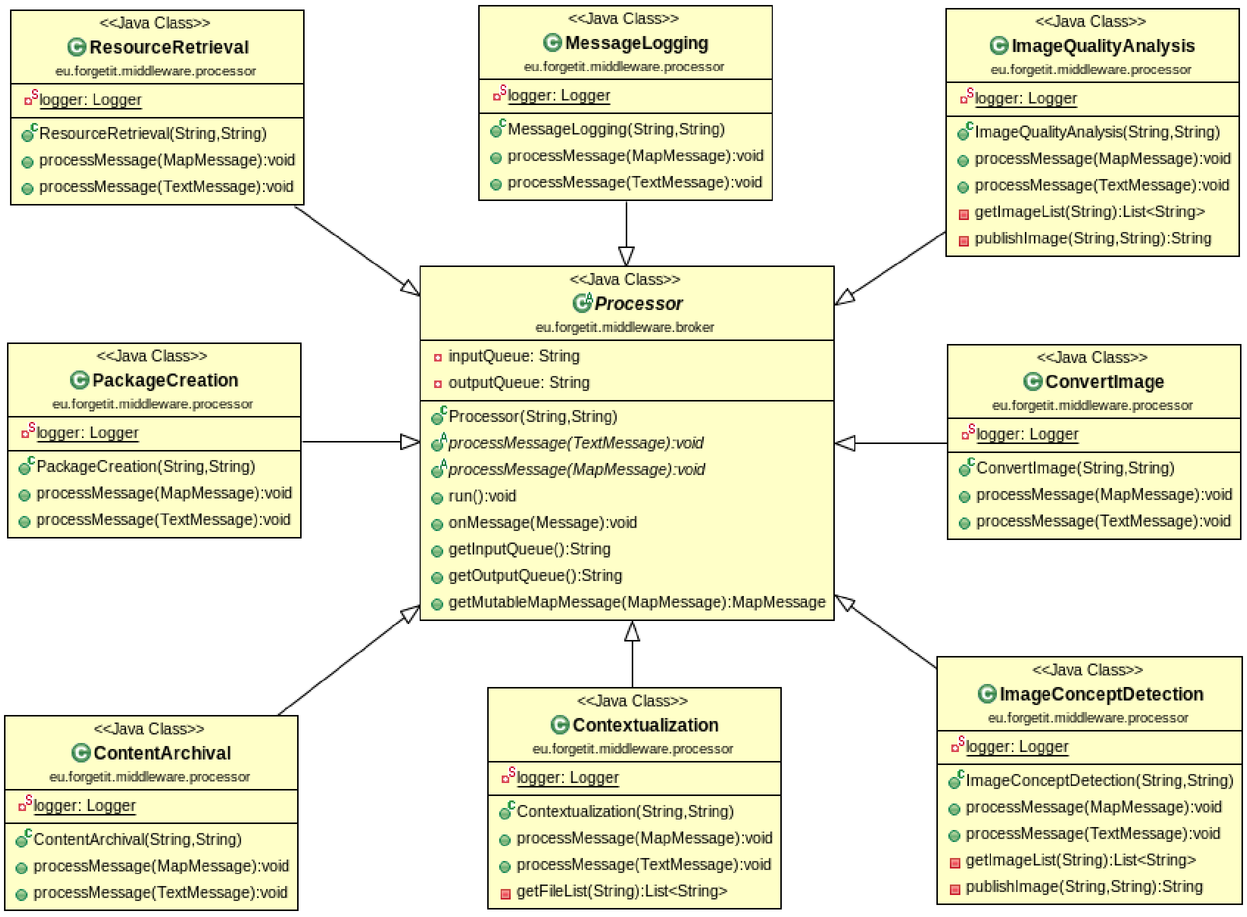
<<Java Class>>
Response
eu.forgetit.middleware.server
entries: HashMap<String,String>
Response()
addValue(String,String):void
getResponse():Map<String,String>
<<Java Class>>
ConfigurationManager
eu.forgetit.middleware
PROPERTY_FILE: String = "config.properties"
ConfigurationManager()
getConfiguration():Configuration
<<Java Enumeration>>
JobType
eu.forgetit.middleware.component
TEST: JobType
SYNERGETIC_PRESERVATION: JobType
JobType()
-response
0..1
-configuration
0..1
<<Java Interface>>
Configuration
org.apache.commons.configuration
<<Java Class>>
Scheduler
eu.forgetit.middleware.component
logger: Logger = LoggerFactory.getLogger(Scheduler.class)
queueName: String = null
message: MapMessage = null
Scheduler()
process(Job):void
<<Java Class>>
SemanticDesktop
eu.forgetit.middleware.server
logger: Logger = LoggerFactory.getLogger(SemanticDesktop.class)
SemanticDesktop()
updatePVGET(String,String,String):Response
updatePV(String,String,String):Response
restore(String):Response
getStatus(String):Response
<<Java Class>>
Job
eu.forgetit.middleware
id: String = UUID.randomUUID().toString()
workflow: String = null
valuesMap: Map<String,String> = new HashMap<String,String>()
Job()
getId():String
getWorkflow():String
setWorkflow(String):void
addValue(String,String):void
getValue(String):String
getValuesMap():Map<String,String>
setValuesMap(Map<String,String>):void
<<Java Class>>
WorkflowManager
eu.forgetit.middleware
logger: Logger = LoggerFactory.getLogger(WorkflowManager.class)
WorkflowManager()
dispatch(Map<String,String>):void

Figure 6: Class diagram for PoF Middleware interface, with associated classes.



<<Java Class>>
MessageLogger
eu.forgetit.middleware.broker
logQueue: String = ConfigurationManager.getConfiguration().getString("log.queue")
MessageLogger()
send(String):void
send(TextMessage):void
<<Java Class>>
Processor
eu.forgetit.middleware.broker
inputQueue: String = null
outputQueue: String = null
Processor(String,String)
processMessage(TextMessage):void
processMessage(MapMessage):void
run():void
onMessage(Message):void
getInputQueue():String
getOutputQueue():String
getMutableMapMessage(MapMessage):MapMessage
<<Java Class>>
QueueManager
eu.forgetit.middleware.broker
logger: Logger = LoggerFactory.getLogger(QueueManager.class)
configuration: Configuration = null
threadList: List<Thread> = null
getInstance():QueueManager
QueueManager()
sendMessage(String,String):void
sendMessage(Message,String):void
getConsumer(String):MessageConsumer
init():void
thread(Runnable,String,boolean):void
stopBrokerThreads():void
-queueManager
0..1
-producer
0..1
<<Java Class>>
ConnectionFactory
eu.forgetit.middleware.broker
logger: Logger = LoggerFactory.getLogger(ConnectionFactory.class)
connectionFactory: ActiveMQConnectionFactory = null
connection: Connection = null
connected: boolean = false
consumers: List<MessageConsumer> = null
ConnectionFactory()
getSession():Session
addConsumer(MessageConsumer):void
checkConnection(String,int):void
closeConnection():void
<<Java Class>>
Producer
eu.forgetit.middleware.broker
logger: Logger = LoggerFactory.getLogger(Producer.class)
session: Session = null
message: Message = null
queue: String = null
Producer(Session,Message,String)
run():void

Figure 7: Class diagram for PoF Middleware broker, with associated classes.



<<Java Class>>
ResourceRetrieval
eu.forgetit.middleware.processor
logger: Logger
ResourceRetrieval(String,String)
processMessage(MapMessage):void
processMessage(TextMessage):void
<<Java Class>>
MessageLogging
eu.forgetit.middleware.processor
logger: Logger
MessageLogging(String,String)
processMessage(MapMessage):void
processMessage(TextMessage):void
<<Java Class>>
ImageQualityAnalysis
eu.forgetit.middleware.processor
logger: Logger
ImageQualityAnalysis(String,String)
processMessage(MapMessage):void
processMessage(TextMessage):void
getImageList(String):List<String>
publishImage(String,String):String
<<Java Class>>
Processor
eu.forgetit.middleware.broker
inputQueue: String
outputQueue: String
Processor(String,String)
processMessage(TextMessage):void
processMessage(MapMessage):void
run():void
onMessage(Message):void
getInputQueue():String
getOutputQueue():String
getMutableMapMessage(MapMessage):MapMessage
<<Java Class>>
PackageCreation
eu.forgetit.middleware.processor
logger: Logger
PackageCreation(String,String)
processMessage(MapMessage):void
processMessage(TextMessage):void
<<Java Class>>
ConvertImage
eu.forgetit.middleware.processor
logger: Logger
ConvertImage(String,String)
processMessage(MapMessage):void
processMessage(TextMessage):void
<<Java Class>>
ContentArchival
eu.forgetit.middleware.processor
logger: Logger
ContentArchival(String,String)
processMessage(MapMessage):void
processMessage(TextMessage):void
<<Java Class>>
Contextualization
eu.forgetit.middleware.processor
logger: Logger
Contextualization(String,String)
processMessage(MapMessage):void
processMessage(TextMessage):void
getFileList(String):List<String>
<<Java Class>>
ImageConceptDetection
eu.forgetit.middleware.processor
logger: Logger
ImageConceptDetection(String,String)
processMessage(MapMessage):void
processMessage(TextMessage):void
getImageList(String):List<String>
publishImage(String,String):String

Figure 8: Class diagram for PoF Middleware Processor, with associated classes.

5 PoF Middleware Integrated Components

In this Section we summarize the status of the components integrated in the first release of the PoF Framework, according to the integration plan reported in Table 15 of deliverable D8.1 [ForgetIT 2013d]. For each component we report the corresponding WPs and the reference deliverables (if any), a short description of the role in the overall architecture and the deployment mechanism for its integration, the main APIs and the development status. We also provide additional references or external links whenever available and preliminary information about the license. The information for each component will be updated and completed in the next PoF Framework deliverables.

5.1 ID Manager

WP and Deliverables WP8 (for integration in the messaging layer), WP3 (for scheduling of forgetting process), WP5 (for scheduling of archiving process).

Description This component mediates between the IDs used in the Preservation System components (Digital Repository and Cloud Storage Service) and the IDs used in the Active Systems. It can also be used to generate new unique IDs. The IDs are managed by means of Enterprise JavaBeans (EJB) objects, with `get` and `set` methods to edit the internal properties corresponding to the given ID. The ID Manager is often invoked internally by the Scheduler (see Figure 9), for assigning new IDs to content processed in the middleware.

Deployment The ID Manager is written in Java and is included in the main PoF Middleware Java project (`eu.forgetit.middleware.component` package). The dependencies are managed with Maven.

API and I/O formats The APIs of the ID Manager are used by all internal components of the middleware, but they are also exposed to the other framework components, namely the Active Systems and the Preservation System. REST APIs are published using Jersey[4], the reference implementation of JAX-RS specification for RESTful web services. The ID Manager provides API for creating new IDs and maintains the mapping among different IDs. Main methods include generation of new ID and retrieval of IDs from a internal repository. The IDs are stored in a pure object database, ObjectDB[5], where CRUD operations are implemented using Java Persistence API. Different standards are available for identifiers, in the current implementation we make use of UUID. The IDs assigned to each resource correspond to the ID in the user application (e.g. the original resource ID created by PIMO), the ID assigned by the Digital Repository (e.g. the DSpace ID) and the ID assigned by the PDS (the name ID used to store the resource and for maintaining different versions). There is also a PoF Middleware ID (UUID) which is created when a preservation request is triggered in the PoF Middleware. The APIs of the ID Manager and

[4]Jersey - `https://jersey.java.net`
[5]ObjectDB - `http://www.objectdb.com`

the associated classes in the middleware are shown in Figure 12

Development status The basic implementation for simple time-based scheduling of processes as well as the integration in the framework is done. The next release will include improvements and extensions of the methods based on the requirements of the other components.

Documentation and reference links Methods and examples are available in the software documentation, see Section 8.

License The component is released under Open Source license, the same used for the PoF Middleware, see Section 8.

5.2 Scheduler

WP and Deliverables Component developed within WP8, the first version is described here, deliverable D8.4 [ForgetIT 2015a] will describe the first complete release.

Description The Scheduler is responsible for managing and organizing middleware activities, by receiving and dispatching requests for the different workflows and processes and by interacting with the messaging infrastructure. Currently the Scheduler triggers preservation workflows, either by receiving input from other PoF Middleware components or by executing scheduled activities. The Scheduler is currently available as a preliminary prototype, written in Java, supporting basic workflows. In the next framework releases will leverage Apache Camel as rule-engine for advanced workflows. The Scheduler receives as input a job created by the WorkflowManager (JobType objects in Figure 9), based on the request received through the REST interface. The Scheduler can extract relevant information from each job and communicate with the messaging layer components, such as the QueueManager (see Figure 9), starting different processes by sending messages to the appropriate middleware queues. An example is provided in Appendix A. The Scheduler depends on the ID Manager (to get the correct IDs associated to content, to be passed in the MapMessage object) and on the ConfigurationManager (to get information about configuration, such as the broker URL or other parameters).

Deployment The Scheduler is written in Java and is included in the main PoF Middleware Java project (`eu.forgetit.middleware.component` package). The dependencies are managed with Maven. The WorkflowManager is an additional component that has been created to bridge the gap between the web server and the messaging layer, preserving loose coupling, but could be removed or embedded in the future releases of the Scheduler when using Apache Camel.

API and I/O formats The Scheduler has to provide APIs that allow the scheduling of processes based on time and events, to request status information and to delete scheduled events. A subset of these APIs has been already implemented for the first release. The Scheduler currently exposes APIs for processing jobs received by the WorkflowManager through the REST web server. The only public method at the moment is `process`, which takes as argument a JobType object (see Figure 9). According to the request type, the Scheduler can trigger different workflows. At the moment a basic synergetic preservation workflow and a resource restore workflow have been implemented. The job representation is based on XML and Java Enterprise JavaBeans (EJB), while the messages adopt the Java Message Service (JMS) specification. The JMS MapMessage structure (see Figure 9) is used to create key-value pairs with information used by other components (e.g. content ID, resource path, etc.)

Development status Early prototype supporting basic workflows is available, to be upgraded to integrate with Apache Camel and to support more complex workflows.

Documentation and reference links Methods and examples are available in the software documentation, see Section 8.

License The component is released under Open Source license, the same used for the PoF Middleware, see Section 8.

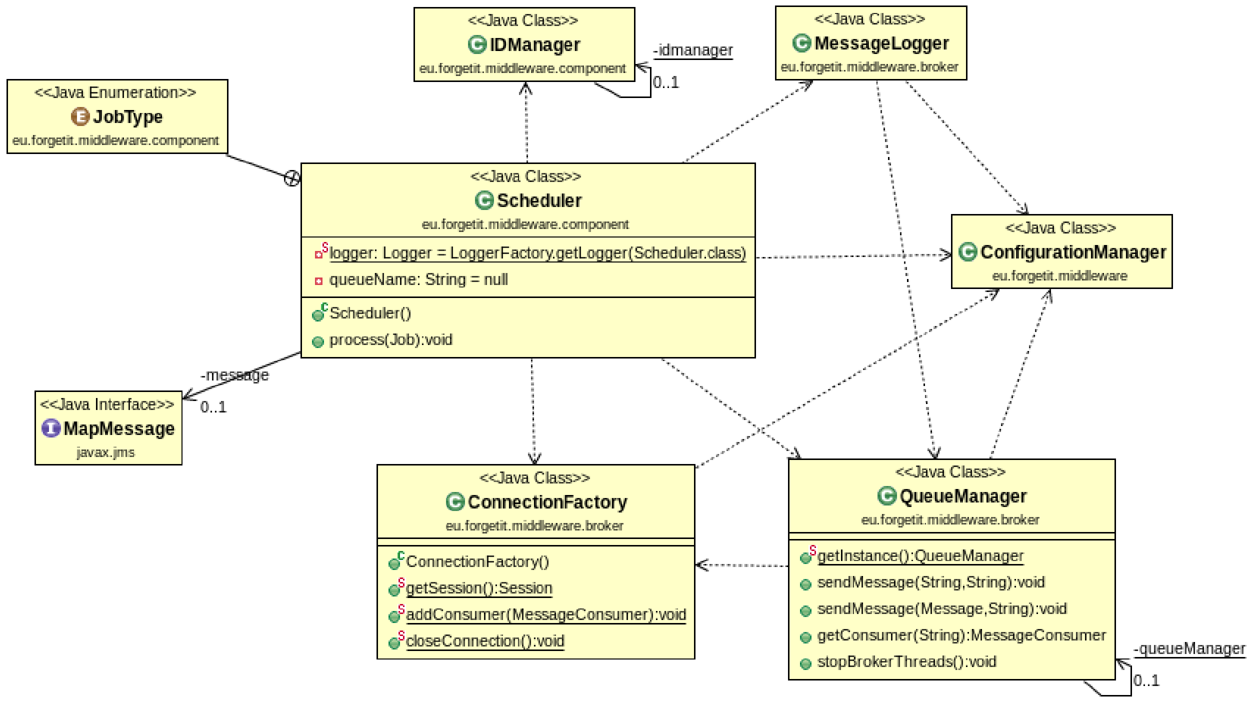


Figure 9: Class diagram for Scheduler component, with associated classes.

5.3 Forgettor

WP and Deliverables D3.2, which discusses the conceptual development and architecture design of Memory Buoyancy Component as part of WP3, is integrated to the middleware.

Description The Memory Buoyancy (MB) Component is responsible for estimating the memory buoyancy values of a resource (Section 5.2, D3.1) in personal preservation contexts. The component relies on the activity history of users in the information space (his semantic desktops), as well as the ontological knowledge of the resource, including its structures and its relationships with other resources. In the middleware layer, the Memory Buoyancy Component is used as a service by the clients (PIMO or TYPO3 systems) to enumerically assess a resource.

Deployment The MB component has two sub-components. The first sub component serves as a background job that periodically gets triggered and estimates the resources' MB values. The results are then cached in a database. The second component, which is deployed directly to the middleware, is a web service repository that dispatches requests about MB values to the database and return results for respective context (time, persons who question, ...).

API and I/O formats The APIs Input/Output format of all services are available on L3S REST server hosting the Forgettor[6].

- *Querying MB Values of PIMO resources*: this service allows the client to query the estimated MB values, and get a numerical value from 0 to 1 in plain-text as a result (or NaN if the values are not yet estimated, or the resource is not registered in the system).
    1. REST service type: GET.
    2. URI Input: `http://forgetit.l3s.uni-hannover.de:8092/pimo/mb/query?u=<userID>&r=<resourceID>&t=<timestampinUNIXepochs>`.
    3. URI output: (plain-text) MB score in [0,1] or NaN.
    4. Query example: `http://forgetit.l3s.uni-hannover.de:8092/pimo/mb/query?u=pimo:1327593979868:1&r=pimo:1381327141334:56&t=1384506130`.

- *Register PIMO resources*: in order to compute the MB scores using the background sub-component, the resources must be registered; this service allows the client to send the list of resource IDs to register for the computation.
    1. REST service type: POST.
    2. URI Input: `http://forgetit.l3s.uni-hannover.de:8092/pimo/res/register`.
    3. URI output: A JSON reponse object that contains:
        - the response status code:
            * CREATED: the resources have been successfully registered.
            * NOT_MODIFIED: the resources are already registered, or the attempt makes no changes in the system.
            * INTERNAL_SERVER_ERROR: the server failed to register due to internal error.

[6]Forgettor Server - `http://forgetit.l3s.uni-hannover.de:8092/application.wadl`

    * PARTIAL_CONTENT (for bulk registration): only a sub set of resources are registered.
  - the list of IDs of successfully registered resources.

4. Example client code in Java:

```
import javax.ws.rs.client.AsyncInvoker;
import javax.ws.rs.client.Client;
import javax.ws.rs.client.ClientBuilder;
import javax.ws.rs.client.Entity;
import javax.ws.rs.client.InvocationCallback;
import javax.ws.rs.client.WebTarget;
import javax.ws.rs.core.MediaType;
import org.glassfish.jersey.client.ClientConfig;
import eu.forgetit.l3s.services.schema.MBRequest;
import eu.forgetit.l3s.services.schema.MBVEntity;
import eu.forgetit.l3s.services.schema.MBVList;

...

// Define the entry point of the web service domain
ClientConfig clientConfig = new ClientConfig();
client = ClientBuilder.newClient(clientConfig);
WebTarget target = client.target("http://forgetit.l3s.uni
-hannover.de:8092");

...

// Define an asynchronous REST request
final AsyncInvoker asyncInvoker = target.path("/pimo/mb/
bulk-query").request(MediaType.APPLICATION_JSON).async();

// Define a request object which contains collection ID
 (account), epoch value of demanded calculation timestamp,
 and a list of resource IDs

MBRequest req = new MBRequest();
req.setAccount("pimo:1327593979868:1");
req.setTime(1386686731);

List<String> res = new ArrayList<>(4);
res.add("pimo:1381327141334:56");
res.add("pimo:1374842706949:3");
res.add("pimo:1385386608202:18");
res.add("pimo:1381327141334:55");
res.add("pimo:1365627012409:41");

req.setResources(res);

// Send the asynchronous request to the service
Entity<MBRequest> reqEntity = Entity.entity(req, MediaType
.APPLICATION_JSON);

MBVList futureResp = null;
try {
    futureResp = asyncInvoker.post(reqEntity, new
    InvocationCallback<MBVList>() {
        @Override
        public void completed(MBVList response) {
            System.out.println("Response entity '" +
            response + "' received.");
            for (MBVEntity mbve : response.getValues()) {
              System.out.println(mbve.toCompiledString());
            }
        }

        @Override
        public void failed(Throwable throwable) {
            System.out.println("Invocation failed.");
            throwable.printStackTrace();
        }
    }).get();
} catch (InterruptedException | ExecutionException e) {
    e.printStackTrace();
}
```

Development status The Forgettor component is currently under development according to the plan in deliverable D8.1 [ForgetIT 2013d]. A new version will be integrated in the next release of the PoF Framework.

Documentation and reference links Additional information about the component can be found in deliverable D3.2 Section 2.2-2.3 [ForgetIT 2014a].

Licenses The different components of the Forgettor are available under GNU License GPL v3.0, Creative Commons License 3 and Apache License 2.0.

5.4 Extractor

WP and Deliverables The Extractor is part of WP4. The different technologies that are required for realizing the Extractor were reviewed in deliverable D4.1 [ForgetIT 2013b]. Detailed description of the initial versions of the developed multimedia analysis methods of the extractor, including usage examples, were presented in deliverable D4.2 [ForgetIT 2014b].

Description The Extractor in its preliminary version, consists of two subcomponents, for image quality assessment and concept detection respectively. Image quality assessment takes as input an image (or a set of images) and returns its visual quality score by examining the presence of visual artifacts such as low contrast, noise, blur, etc. Concept detection calculates the confidence scores for a set of concepts which indicate how much each concept is related to the image. It takes as input an image (or a set of images) and returns for each image a vector that contains the confidence scores for all the concepts. The class diagram of the Extractor is shown in Figure 10, where the associated Processor in the messaging layer is also shown.

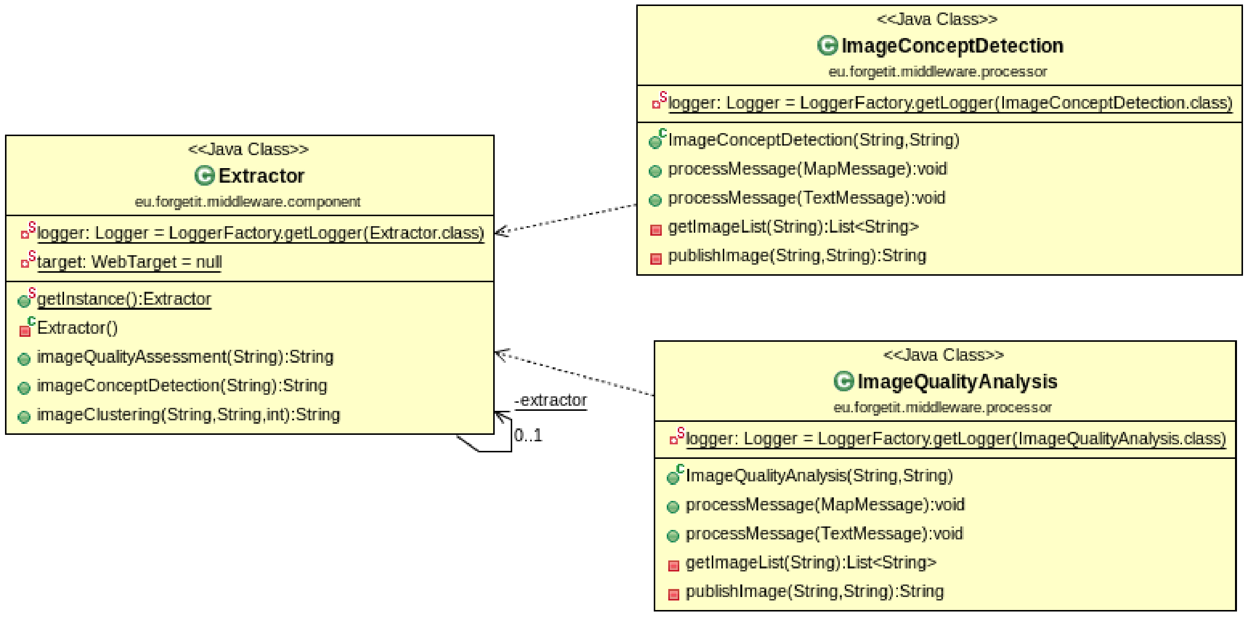
Deployment Both subcomponents of the preliminary version of the Extractor have been deployed as REST services running in CERTH's servers.

API and I/O formats APIs and response format (XML-based) is documented in deliverable D4.2 [ForgetIT 2014b].

Development status The image quality assessment subcomponent will be extended in order to assess the aesthetic quality of an image. In concept detection, the concepts set will be modified in order to include more representative concepts and the effectiveness of the automatic annotation will be improved. Furthermore, more subcomponents will be added in the next version of the Extractor including image and face clustering, fast detection of duplicate images, single/multidocument text summarization and automatic entity disambiguation in text.

Documentation and reference links Additional information about the Extractor component and the RESTful web service can be found in deliverable D4.2 [ForgetIT 2014b].

License The Extractor subcomponents are Copyright ©2013-2014 CERTH. Some of these subcomponents make internal use of third party software and libraries, such as OpenCV (BSD license) and Liblinear (Copyright ©2007-2013 the LIBLINEAR Project). The license for the Extractor component will be further analysed and described in deliverable D8.4 [ForgetIT 2015a].



<<Java Class>>
Extractor
eu.forgetit.middleware.component
logger: Logger = LoggerFactory.getLogger(Extractor.class)
target: WebTarget = null
getInstance():Extractor
Extractor()
imageQualityAssessment(String):String
imageConceptDetection(String):String
imageClustering(String,String,int):String
-extractor
0..1
<<Java Class>>
ImageConceptDetection
eu.forgetit.middleware.processor
logger: Logger = LoggerFactory.getLogger(ImageConceptDetection.class)
ImageConceptDetection(String,String)
processMessage(MapMessage):void
processMessage(TextMessage):void
getImageList(String):List<String>
publishImage(String,String):String
<<Java Class>>
ImageQualityAnalysis
eu.forgetit.middleware.processor
logger: Logger = LoggerFactory.getLogger(ImageQualityAnalysis.class)
ImageQualityAnalysis(String,String)
processMessage(MapMessage):void
processMessage(TextMessage):void
getImageList(String):List<String>
publishImage(String,String):String

Figure 10: Class diagram for Extractor, with associated classes.

5.5 Contextualizer

WP and Deliverables The Contextualizer is part of WP6. The different technologies that are required for realizing the Contextualizer were reviewed in D6.1. Detailed description of prototype versions of a number of contextualization components, including usage examples, were presented in deliverable D6.2 [ForgetIT 2014d].

Description There are currently a number of components developed to perform contextualization over both text and images. The image component contextualizes a photo collection by adding additional images which provide a richer context. In contrast the text components add context in a number of different ways including the addition of full text and ontology instances. The class diagram for the Contextualizer is shown in Figure 11, where the associated Processor in the messaging layer is also shown.

Deployment The prototype version of the contextualization via disambiguation component has been deploayed as a REST service integrated into the PoF Middleware.

API and I/O formats The integrated component is accessed via a REST interface.

Documentation and reference links Additional information about the Contextualizer is available in deliverable D6.2 [ForgetIT 2014d].

License The contextualization via disambiguation component is Copyright ©2013-2014 The University of Sheffield and is released under the LGPL.

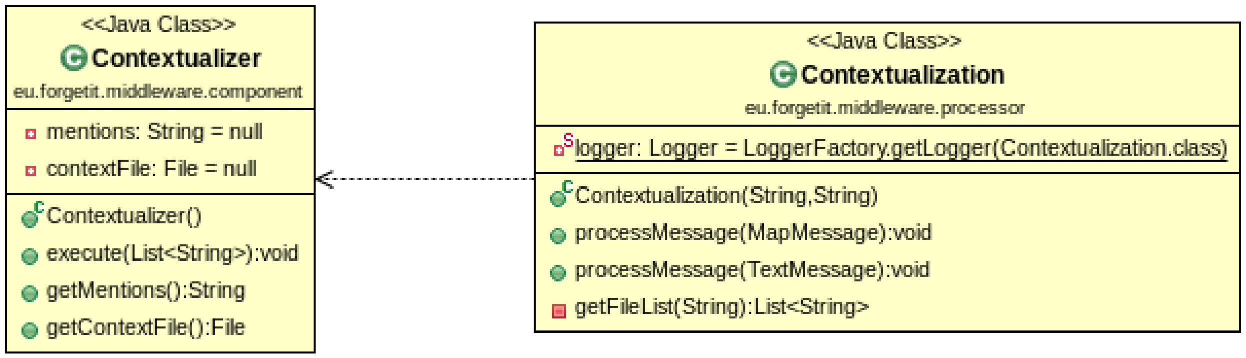

<<Java Class>>
Contextualizer
eu.forgetit.middleware.component
mentions: String = null
contextFile: File = null
Contextualizer()
execute(List<String>):void
getMentions():String
getContextFile():File
<<Java Class>>
Contextualization
eu.forgetit.middleware.processor
logger: Logger = LoggerFactory.getLogger(Contextualization.class)
Contextualization(String,String)
processMessage(MapMessage):void
processMessage(TextMessage):void
getFileList(String):List<String>

Figure 11: Class diagram for Contextualizer, with associated classes.

5.6 Collector/Archiver

WP and Deliverables Component developed in WP5, described in deliverable D5.2 [ForgetIT 2014c].

Description The Collector is responsible for fetching resources from user applications using CMIS, to create the SIP structure and to build the SIP collecting the results from all components. Archiver is responsible for ingesting SIP into Preservation System. Resources in the Preservation System can be retrieved using the Collector, which interacts with the ID Manager to get information about the resource IDs, as shown in Figure 12.

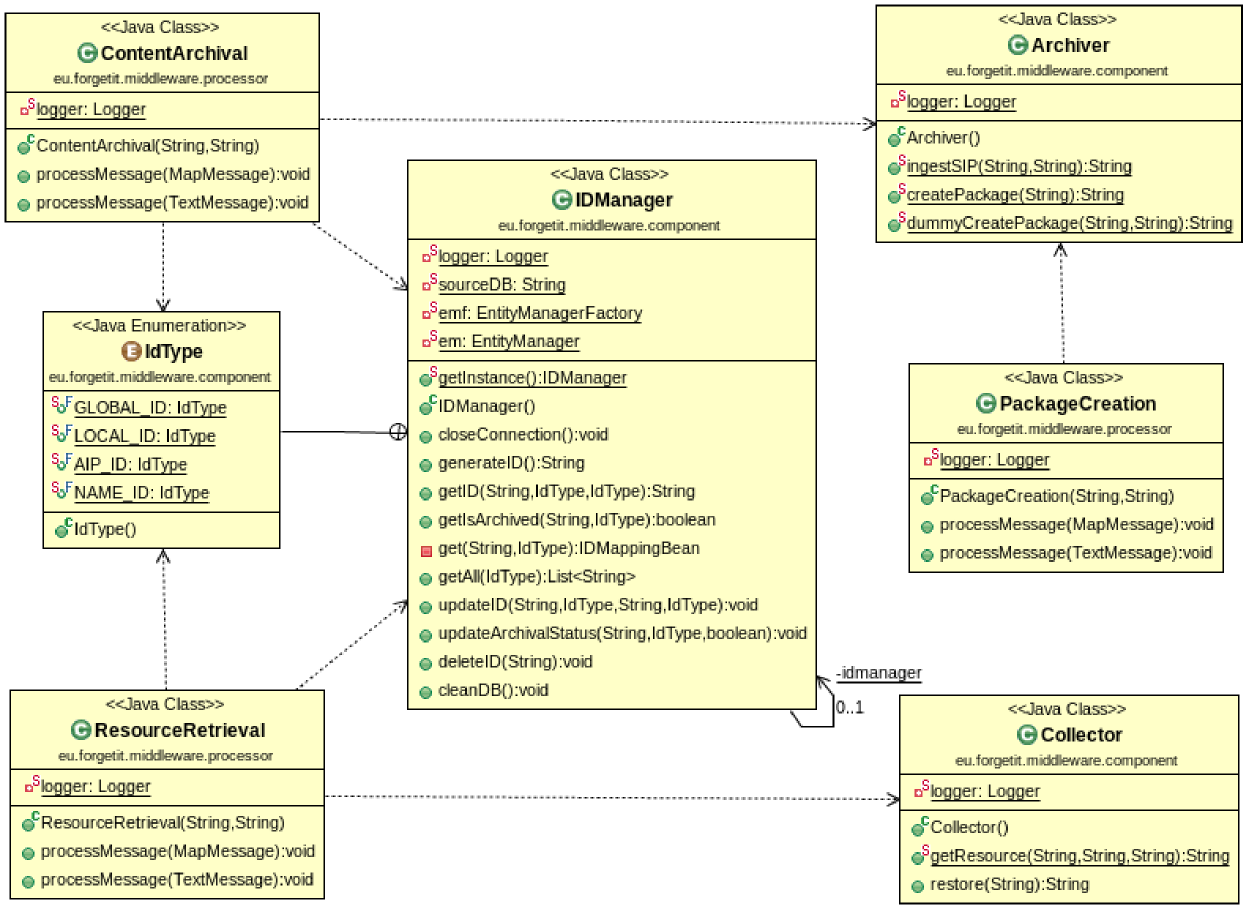
Deployment Currently the component is available as Java Archive added as additional dependency to the PoF Middleware Java project. In the future the component source code could be integrated in the `eu.forgetit.middleware.component` package.

API and I/O formats The component provides APIs for fetching content from user applications using CMIS protocol and to manage SIP/DIP.

Development status Preliminary prototype, used for implementing the two basic workflows. Requires further testing and development to support advanced features related to concurrency and to better implement the CMIS client.

Documentation and reference links Available in D5.2.

License Available as open source license from LTU.



<<Java Class>>
ContentArchival
eu.forgetit.middleware.processor
logger: Logger
ContentArchival(String,String)
processMessage(MapMessage):void
processMessage(TextMessage):void
<<Java Class>>
Archiver
eu.forgetit.middleware.component
logger: Logger
Archiver()
ingestSIP(String,String):String
createPackage(String):String
dummyCreatePackage(String,String):String
<<Java Class>>
IDManager
eu.forgetit.middleware.component
logger: Logger
sourceDB: String
emf: EntityManagerFactory
em: EntityManager
getInstance():IDManager
IDManager()
closeConnection():void
generateID():String
getID(String,IdType,IdType):String
getIsArchived(String,IdType):boolean
get(String,IdType):IDMappingBean
getAll(IdType):List<String>
updateID(String,IdType,String,IdType):void
updateArchivalStatus(String,IdType,boolean):void
deleteID(String):void
cleanDB():void
<<Java Enumeration>>
IdType
eu.forgetit.middleware.component
GLOBAL_ID: IdType
LOCAL_ID: IdType
AIP_ID: IdType
NAME_ID: IdType
IdType()
<<Java Class>>
PackageCreation
eu.forgetit.middleware.processor
logger: Logger
PackageCreation(String,String)
processMessage(MapMessage):void
processMessage(TextMessage):void
-idmanager
0..1
<<Java Class>>
ResourceRetrieval
eu.forgetit.middleware.processor
logger: Logger
ResourceRetrieval(String,String)
processMessage(MapMessage):void
processMessage(TextMessage):void
<<Java Class>>
Collector
eu.forgetit.middleware.component
logger: Logger
Collector()
getResource(String,String,String):String
restore(String):String

Figure 12: Class diagram for ID Manager and Collector/Archiver, with associated classes.

6 Active Systems

6.1 Semantic Desktop

WP and Deliverables Main development in WP9 with technical contributions from WP3, WP4, WP5, and WP8. Explanations in deliverables D8.1 Section 3.1 [ForgetIT 2013d], D9.1 [ForgetIT 2013e] and especially D9.2 [ForgetIT 2014g], when describing the mock-ups.

Description The Semantic Desktop is a personal information management system with an underlying ontology semantically describing the users mental model and the resources involved. This ontology is the Personal Information MOdel (PIMO). The Semantic Desktop infrastructure consists of a PIMO Server with a dedicated API so that any third party software could use the PIMO for its 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 users 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.

Deployment The PIMO consists of a number of web applications that are deployed (as WAR files) to an Apache Tomcat application server running on a dedicated virtual machine in the EURIX testbed environment.

API and I/O formats In the ForgetIT context, the PIMO provides its contents through a CMIS service. It uses a SemanticDesktop endpoint on the middleware (see Figure 6) to set preservation values for documents and to restore documents from the archive.

Development status All functionality for the basic workflow is implemented. This includes the CMIS service as well as the connection to the middleware REST API. In further releases, there will be a more complex communication with the REST API.

Documentation and reference links Additional information about the PIMO is available on DFKI PIMO web site[7].

Licence The componenent is available under a BSD-compliant license for interfaces and PIMO model; the implementation is available for free use in ForgetIT under the consortium agreement terms.

[7]DFKI PIMO - `https://pimo.opendfki.de`

6.2 TYPO3

WP and Deliverables Main development in WP10 with conceptual contribution from WP2 and technical from WP3, WP4, WP5 and WP8. The architecture of the overall system structure is described in D8.1. Details about how TYPO3 CMS will be amended and integrated can be found in D10.1 [ForgetIT 2013a], including mock-ups.

Description TYPO3 CMS is an open source web Content Management System, used by a wide range of organizations around the globe. It is written in PHP and fully open sourced under GPL/MIT licenses. In the context of the ForgetIT, dkd will develop a set of TYPO3 extensions which will allow usage of the PoF Framework and the datastore. The scope of those extensions is described in D10.1 [ForgetIT 2013a].

Deployment TYPO3 CMS runs on a web server using MySQL and PHP. To connect to the PoF Framework a second server running the Java application Alfresco Community Edition (CE) is used as a CMIS content repository. Both systems are pre-configured and set up as dedicated virtual machines in the EURIX testbed environment.

API and I/O formats In the ForgetIT context, the TYPO3 CMS provides its contents through a CMIS service. It will utilize REST APIs of the middleware communicate to the PoF Framework.

Development status During the first year, dkd delivered a description of the concepts/technical solution design in D10.1 [ForgetIT 2013a] and a proof of concept to verify TYPO3 CMS integration into CMIS. The plan for the second year is to deliver a first running, releasable version including semantic annotation. Currently dkd is working on a scalable integration of CMIS into TYPO3 CMS by creating a set of five extensions.

Documentation and reference links For more background on TYPO3 visit the project web site[8]. For additional information about Alfresco CE, visit the product web site[9].

Licence The licences have not been chosen yet, but they will likely be either GPL or Apache compliant licenses, depending on the code dkd bases its work on.

[8]TYPO3 - `http://www.typo3.org`

[9]Alfresco Community Edition - `http://www.alfresco.com/products/community`

7 Preservation System

In the following Sections we describe the implementation of the two components of the Preservation System, namely the Digital Repository and the Cloud Storage Service.

7.1 Digital Repository: DSpace

As described in D8.1 [ForgetIT 2013d], several candidate platforms have been evaluated for the Digital Repository. DSpace[10] has been selected according to the identified assessment criteria. DSpace digital repository is already described in detail in Section 7.1 and Table 18 of deliverable D8.1 [ForgetIT 2013d]. In order to enable the interaction between the PoF Middleware and the DSpace repository, REST APIs for both the access and the ingest processes have been implemented. The ingest interface is used to trigger operations related to the SIP validation, its submission and the creation of the AIP. The access interface is used for the dissemination of the AIPs.

```
@Path("access")
public class Access {

        private Response response = null;

        @GET
        @Produces(MediaType.APPLICATION_JSON)
        @Path("configuration")
        public Response showProperties(){...

        @GET
        @Produces(MediaType.APPLICATION_JSON)
        @Path("aip")
        public Response retrieveAIP(@QueryParam("aipID") String
        aipID, @QueryParam("name") String aipName, @QueryParam
        ("format") String format){...
```

```
@Path("ingest")
public class Ingest {

        private Logger logger = LoggerFactory.getLogger(Ingest.
                                                                class);

        private Response response = null;
        private String error = null;

        @POST
        @Consumes(MediaType.MULTIPART_FORM_DATA)
        @Produces(MediaType.APPLICATION_JSON)
        @Path("sip")
        public Response ingestPackage(
        @FormDataParam("file") FormDataBodyPart body,
        @FormDataParam("name") String sipName,
        @FormDataParam("user") String user,
        @FormDataParam("file") FormDataContentDisposition
        fileDetail,
        @Context HttpHeaders headers){...
```

[10]DSpace - http://www.dspace.org

An excerpt of the code of both the ingest and access APIs is reported above, where the Jersey resources exposing REST APIs are shown. For additional details please refer to the software documentation reported in Section 8.2. DSpace internal data model is represented in Figure 13. Digital objects are organized into several layers such as collections, communities, items, and sites. As far as the installation process is concerned, we provide an installation guide tailored to ForgetIT environment in Section B.2 of Appendix B. Additional information can be found in the official DSpace guide on the project wiki[11].

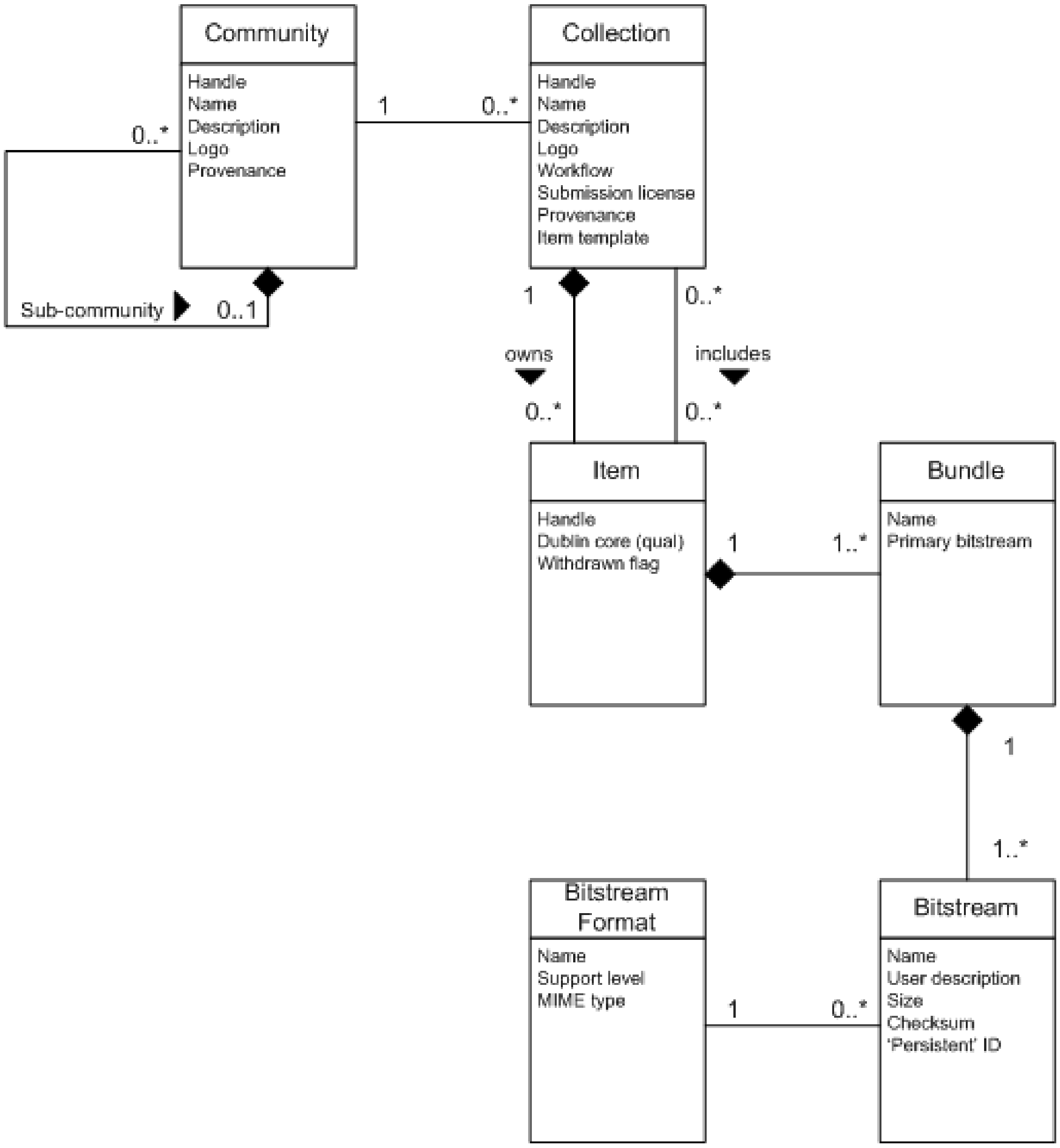
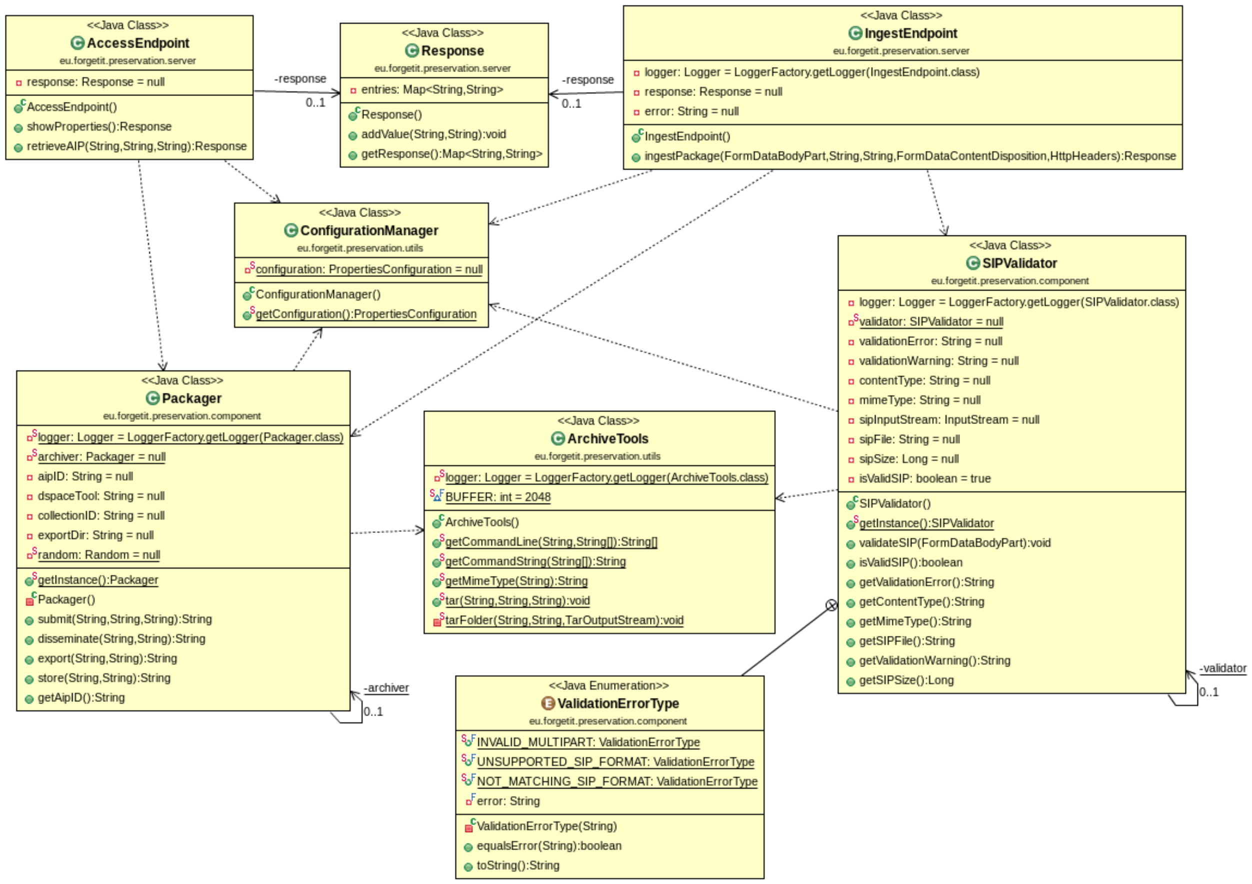


Figure 13: DSpace Data Model diagram.

The ingest and access endpoints exposed by the Preservation System are depicted in Figure 14. The code is written in Java and is deployed in the main PoF Middleware Java

[11]DSpace Guide - `https://wiki.duraspace.org/display/DSDOC4x/Installing+DSpace`

project, as part of the `eu.forgetit.preservation.server` package. A Response class is defined, which enables the conversion of the original server response into different formats (XML, JSON, etc.). Other components of the Digital Repository are shown in Figure 14, for validating the ingested SIP and for performing different operations, including the export of the AIP in the Cloud Storage Service (Packager).



<<Java Class>>
AccessEndpoint
eu.forgetit.preservation.server
response: Response = null
AccessEndpoint()
showProperties():Response
retrieveAIP(String,String,String):Response
-response
0..1
<<Java Class>>
Response
eu.forgetit.preservation.server
entries: Map<String,String>
Response()
addValue(String,String):void
getResponse():Map<String,String>
-response
0..1
<<Java Class>>
IngestEndpoint
eu.forgetit.preservation.server
logger: Logger = LoggerFactory.getLogger(IngestEndpoint.class)
response: Response = null
error: String = null
IngestEndpoint()
ingestPackage(FormDataBodyPart,String,String,FormDataContentDisposition,HttpHeaders):Response
<<Java Class>>
ConfigurationManager
eu.forgetit.preservation.utils
configuration: PropertiesConfiguration = null
ConfigurationManager()
getConfiguration():PropertiesConfiguration
<<Java Class>>
SIPValidator
eu.forgetit.preservation.component
logger: Logger = LoggerFactory.getLogger(SIPValidator.class)
validator: SIPValidator = null
validationError: String = null
validationWarning: String = null
contentType: String = null
mimeType: String = null
sipInputStream: InputStream = null
sipFile: String = null
sipSize: Long = null
isValidSIP: boolean = true
SIPValidator()
getInstance():SIPValidator
validateSIP(FormDataBodyPart):void
isValidSIP():boolean
getValidationError():String
getContentType():String
getMimeType():String
getSIPFile():String
getValidationWarning():String
getSIPSize():Long
-validator
0..1
<<Java Class>>
Packager
eu.forgetit.preservation.component
logger: Logger = LoggerFactory.getLogger(Packager.class)
archiver: Packager = null
aipID: String = null
dspaceTool: String = null
collectionID: String = null
exportDir: String = null
random: Random = null
getInstance():Packager
Packager()
submit(String,String,String):String
disseminate(String,String):String
export(String,String):String
store(String,String):String
getAipID():String
-archiver
0..1
<<Java Class>>
ArchiveTools
eu.forgetit.preservation.utils
logger: Logger = LoggerFactory.getLogger(ArchiveTools.class)
BUFFER: int = 2048
ArchiveTools()
getCommandLine(String,String[]):String[]
getCommandString(String[]):String
getMimeType(String):String
tar(String,String,String):void
tarFolder(String,String,TarOutputStream):void
<<Java Enumeration>>
ValidationErrorType
eu.forgetit.preservation.component
INVALID_MULTIPART: ValidationErrorType
UNSUPPORTED_SIP_FORMAT: ValidationErrorType
NOT_MATCHING_SIP_FORMAT: ValidationErrorType
error: String
ValidationErrorType(String)
equalsError(String):boolean
toString():String

Figure 14: Class diagram for Preservation System endpoint, with associated classes.

7.2 Cloud Storage: PDS and Storlet Engine

Despite the increase in the ability to store digital data, the ability to maintain these data readable and useful over time decreases, for example, due to frequent changes in rendering technologies. Preservation DataStores (PDS) [Rabinovici-Cohen u. a. 2008; 2013] component provides a preservation-aware storage infrastructure for ForgetIT on top of OpenStack cloud storage. PDS supports the core standard for digital preservation systems, namely the Open Archival Information System (OAIS) [CCSDS 2012] and provides functions for the storage and retrieval of AIPs. PDS exposes an HTTP-based external interface that is based on OAIS and supports operations for ingest, access and preservation actions of AIPs.

PDS performs preservation-related computations functions within the storage system via Storlets running in a sandbox. Storlets are general purpose (yet restricted) routines that may execute computations close to the data within the storage layer. Running within the storage close to the data, Storlets can reduce the bandwidth required to move bytes to an application server (possibly over WAN) for processing. Storlets also improve security and reduce the exposure of private data over the network.

PDS works on top of OpenStack Swift[12], an open source software for creating redundant, fault-tolerant, eventually consistent object storage. It is a distributed scalable system, and uses clusters of standardized servers to store petabytes of accessible data. Having no central brain or master point of control provides greater scalability, redundancy and permanence. Objects are written to multiple hardware devices in the data center, with the OpenStack software responsible for data replication and integrity across the cluster.

The gap analysis of existing storage clouds [Rabinovici-Cohen u. a. 2013] revealed that simply storing data onto the cloud is not an adequate solution for digital preservation repositories. The Archival Storage component is designed to overcome some of these gaps. For example, a digital preservation system may need to perform periodical data intensive tasks, such as data transformations. Data transformation computations are typically data intensive; hence, they are natural candidates to be executed as Storlets, close to the data stored on the same local server. The typical purpose for performing data transformation Storlets includes ease of future use (e.g., the target format is easier to read and render than the original format), efficiency gain (e.g., the target format consumes less space to store, or less CPU cycles to render), and change of standards over time.

One data transformation Storlet that has been already integrated and tested in the PoF Framework is a Storlet for automatically converting proprietary-formatted DNG images to non-proprietary TIFF files, for viewing with general purpose applications. This Storlet uses the ImageMagick[13] open source library.

Another Storlet which has been implemented is based on an image analysis software package, which was prepared by ForgetIT partners in WP4, see deliverable D4.2 [ForgetIT 2014b]. The Storlet runs on a set of image objects, and uses the image analysis software package to analyze them, and extracts feature vectors, which may be used to characterize and describe the input images.

[12]OpenStack Swift - `https://wiki.openstack.org/wiki/Swift`
[13]ImageMagick - `http://www.imagemagick.org/script/index.php`

8 First Prototype Implementation

In this Section we provide a quick description of the adopted approach for software development, deployment and testing and also provide a few details about the software documentation and the source code access.

8.1 Software Development, Deployment and Testing

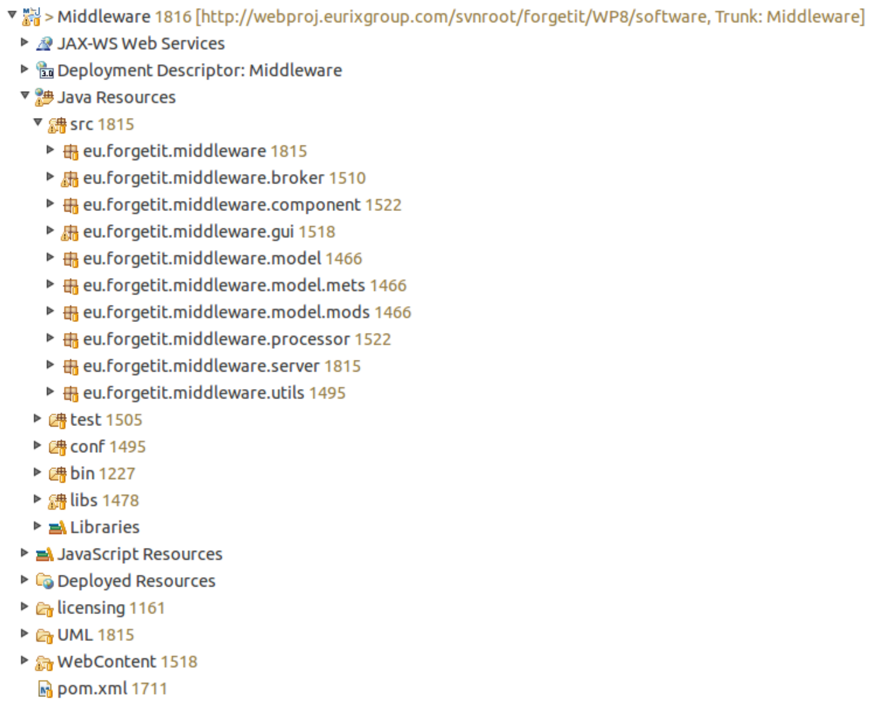
The prototype has been developed adopting the Java language. Two main projects have been created, one for the PoF Middleware and one for the Preservation System. Since the applications in the framework are distributed, we use the Java Enterprise Edition (EE) framework and the Eclipse IDE[14] for Java EE Developers. The version of the IDE used for the first release is *Kepler*, the development environment and the dependencies will be migrated to new versions when they are available. Eclipse is an extensible IDE supporting add-ons and plug-ins developed by the Eclipse community or by third parties. We adopted exclusively open source plug-ins, in order to prevent vendor lock-in for future adopters of ForgetIT software. The software development leverages versioning tools, such as Subversion (SVN)[15], and issue tracking systems such as Trac[16]. We configured Eclipse with additional plug-ins for SVN and Maven (see below). The code repository and the ticketing system are available to all partners and are used by all WPs to establish a collaborative approach. Automatic notification systems for both code changes and issue tracking are used. We use the popular Maven[17] tool to compile, build and deploy the software projects. This tool is integrated in the Eclipse IDE. The external dependencies are written in a `pom.xml` file. The main advantage is that all dependencies are retrieved from public repositories before compiling the software and are included in the software project. The dependency management is simplified and facilitates the work of the development team.

The two main projects for the PoF Middleware and the Digital Repository are Java web applications and their deployment is performed by simply including the *war* files in the `webapps` folder of Tomcat7[18]. When Tomcat starts, all the web applications are deployed and become available for the user. Currently there are three different web applications for the Digital Repository, for the PoF Middleware REST web server and for the PoF Middleware messaging system. The structure of the Java EE projects ib Eclipse for the PoF Middleware and the Preservation System are shown in Figure 15 and Figure 16. The main Java packages of both projects are briefly described in Table 1.

In order to test the developed software, we performed unitary tests using JUnit[19]. Virtualization is used for deploying into the testbed environment the VMs with the components

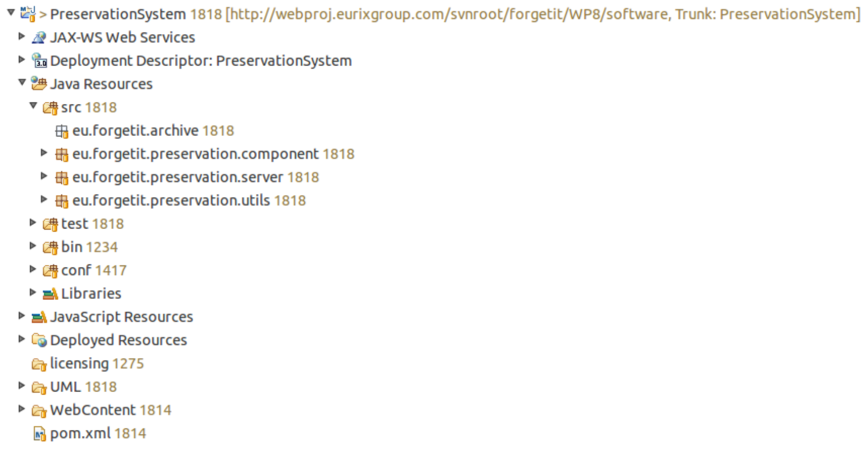
[14]Eclipse IDE - `http://www.eclipse.org`
[15]Apache Subversion - `https://subversion.apache.org`
[16]The Trac Project - `http://trac.edgewall.org`
[17]Apache Maven - `http://maven.apache.org`
[18]Apache Tomcat - `http://tomcat.apache.org`
[19]JUnit - `http://junit.org`

developed by different WPs running on different servers. For the testbed environment we use KVM (see deliverable D8.1[ForgetIT 2013d]), while for smaller test activities VirtualBox[20] has been often used. The component themselves have been verified to work properly through dry run experiments, then test interfaces have been implemented and the interaction between the PoF Middleware and the other components have been tested running different workflows.



```
> Middleware 1816 [http://webproj.eurixgroup.com/svnroot/forgetit/WP8/software, Trunk: Middleware]
  JAX-WS Web Services
  Deployment Descriptor: Middleware
  Java Resources
    src 1815
      eu.forgetit.middleware 1815
      eu.forgetit.middleware.broker 1510
      eu.forgetit.middleware.component 1522
      eu.forgetit.middleware.gui 1518
      eu.forgetit.middleware.model 1466
      eu.forgetit.middleware.model.mets 1466
      eu.forgetit.middleware.model.mods 1466
      eu.forgetit.middleware.processor 1522
      eu.forgetit.middleware.server 1815
      eu.forgetit.middleware.utils 1495
    test 1505
    conf 1495
    bin 1227
    libs 1478
    Libraries
  JavaScript Resources
  Deployed Resources
  licensing 1161
  UML 1815
  WebContent 1518
  pom.xml 1711
```

Figure 15: Structure of the Java EE project for the PoF Middleware



```
> PreservationSystem 1818 [http://webproj.eurixgroup.com/svnroot/forgetit/WP8/software, Trunk: PreservationSystem]
  JAX-WS Web Services
  Deployment Descriptor: PreservationSystem
  Java Resources
    src 1818
      eu.forgetit.archive 1818
      eu.forgetit.preservation.component 1818
      eu.forgetit.preservation.server 1818
      eu.forgetit.preservation.utils 1818
    test 1818
    bin 1234
    conf 1417
    Libraries
  JavaScript Resources
  Deployed Resources
  licensing 1275
  UML 1818
  WebContent 1814
  pom.xml 1814
```

Figure 16: Structure of the Java EE project for the Preservation System

[20]VirtualBox - `https://www.virtualbox.org`

Package	Description
`eu.forgetit.middleware`	This package contains classes to perform basic functions such as managing the workflows, reading properties files, creating key-value maps for the jobs, .
`eu.forgetit.middleware.broker`	The functions to manage the ActiveMQ broker are contained. By means of the `ConnectionFactory` class a JMS Session and a Connection are created when necessary, there are specific classes to create `Processor` and `Producer` objects. There are also a `QueueManager` class for the management of the message queues and a class for the logging messages.
`eu.forgetit.middleware.component`	This package contains a class for each of the PoF Middleware components, for example `Archiver`, `Condensator`, `Forgettor`, and the like.
`eu.forgetit.middleware.gui`	It contains the `MessageReader` servlet class to read incoming messages, there is also the `XMLReader` class whose task is to read and get information from an XML file.
`eu.forgetit.middleware.model`	The `DBManager` class contains methods to interact with the local database. The `IDMappingBean` is used to generate and assign IDs. The classes in these packages corresponding to specific metadata schema, such as Dublin Core and MODS, have been automatically generated from the metadata XML files by means of JAXB.
`eu.forgetit.middleware.processor`	The processing of the message content is carried out through the method of the classes here contained. Such classes are, for example, `ContentArchival`, `ImageConceptDetection` or `PackageCreation`.
`eu.forgetit.middleware.server`	It contains classes to interact with the PoF Middleware such as the `ServiceEndpoint` and the `MiddlewareListener` classes.
`eu.forgetit.middleware.utils`	Several tools for the management of messages and content are provided. For example the `ImageTools` class contains methods to convert images or the `MiddlewareTools` class provides methods to compress a directory (used for packaging purposes).
`eu.forgetit.preservation.component`	`Packager` and `SIPValidator` classes allow to perform the ingest and the validation of a submission package. It is also possible to export an ingested AIP.
`eu.forgetit.preservation.server`	The Digital Repository uses the classes and methods in this package to perform the Ingest and Access operations.
`eu.forgetit.preservation.utils`	It contains classes and methods necessary for the Digital Repository management such as the `ConfigurationManager` class and the `ArchiveTools`, whose methods allow to both create and extract compressed archives.

Table 1: Packages of the PoF Middleware and Preservation System projects.

8.2 Source Code Documentation, Availability and License

The source code of the PoF Framework components and the documentation is available on the project web site along with other project results, at the following URL:

`http://www.forgetit-project.eu/en/project-results`

The pre-compiled binaries (web applications, executables, libraries) as well as instructions for the installation and usage are provided by project partners. When new versions of the framework components are released, they will be updated on the web site. For detailed documentation about each component please refer to WP8 deliverables and to deliverables provided by the corresponding WP.

The project consortium agreed upon releasing the PoF Framework source code under an open source license. The chosen license type (e.g. GPL, LGPL, Apache, etc.) is still under discussion and will be defined for the second prototype release, described in deliverable D8.4 [ForgetIT 2015a].

It is worth noting that the core libraries for the implementation of the PoF Middleware are already available under the Apache license and that several components developed within the project will be released as open source (see previous Sections). Any additional code developed to implement the PoF Middleware will be available as open source, as well. Concerning the Preservation System, the Digital Repository is based on DSpace (available under the BSD license), while the licensing mechanism adopted by IBM for the Storlet Engine is still under evaluation at the moment of writing (Openstack Swift is already available as open source). Concerning the Active Systems, TYPO3 is already available as open source, the licensing of additional customization is still under evaluation, while the Semantic Desktop will be available as open source. A task force has been established in the project, which will evaluate different open source licenses available, also taking into account third party dependencies used to implement the different components.

9 Summary and Future Work

The document provides a description of the first release of the PoF Framework which was demonstrated at the first annual project review. The overall structure of the framework, the way the main components are implemented and how they interact is discussed. The concept of MOM and EIP have been also described. Finally we provided additional details about the software development and documentation.

The development and integration status of the components described in this document is referred to the first release, according to the plan reported in Table 15 of D8.1. The second release of the PoF Framework will be described in deliverable D8.4.

The software prototype reported in this document is the result of the effort performed by all partners during the first year, starting from the architecture defined in deliverable D8.1. The main challenges were the setup of a collaborative environment to share developed software and for testing the results, the agreement on tools and technologies for the development and the integration of the software developed by other partners in a coherent framework. The collaborative tools described in the document and the periodic discussions during WP8 and plenary meetings were very useful for the delivery of the first prototype in time for the first review, anticipating the original plan in the DoW. The same approach will be kept for the second release. A huge effort was required to integrate many components which were still early prototypes. The clear definition of the priority workflows for the first prototype was useful to correctly focus the development effort. New and more complex workflows have been identified for the second release.

Glossary

EAI Enterprise Application Integration. 9, 15

EIP Enterprise Integration Patterns. 6, 9, 15, 61

EJB Enterprise JavaBeans. 22, 24

ESB Enterprise Service Bus. 6, 9

JMS Java Message Service. 24, 59

MB Memory Buoyancy. 24, 25

MOM Message Oriented Middleware. 6, 10, 11, 13, 61

OAIS Open Archival Information System. 6, 35

PDS Preservation DataStores. 11, 35

PIMO Personal Information MOdel. 31

PoF Preserve-or-Forget. 6–11, 22–24, 29, 32, 34, 37, 38, 45, 52, 61

References

[CCSDS 2012] CCSDS: *Reference Model for an Open Archival Information System (OAIS) - Recommended Practice, CCSDS 650.0-M-2 (Magenta Book) Issue 2. Also available as ISO Standard 14721:2012*. `http://public.ccsds.org/publications/archive/650x0m2.pdf`. June 2012. – Retrieved 29 August 2014

[Chappell 2004] CHAPPELL, David: *Enterprise service bus*. O'Reilly Media, Inc., 2004

[DAI und ZHU 2010] DAI, Jun ; ZHU, Xiao-Min: Design and Implementation of an Asynchronous Message Bus Based on ActiveMQ. In: *Computer Systems & Applications* 8 (2010), S. 062

[ForgetIT 2013a] FORGETIT: *Deliverable D10.2: Application Mockups and Prototypes*. February 2013

[ForgetIT 2013b] FORGETIT: *Deliverable D4.1: Information Analysis, Consolidation and Concentration for Preservation – State of the Art and Approach*. July 2013

[ForgetIT 2013c] FORGETIT: *Deliverable D7.1: Foundations of Computational Storage Services*. July 2013

[ForgetIT 2013d] FORGETIT: *Deliverable D8.1: Integration Plan and Architectural Approach*. December 2013

[ForgetIT 2013e] FORGETIT: *Deliverable D9.1: Application Use Cases & Requirements Document*. August 2013

[ForgetIT 2014a] FORGETIT: *Deliverable D3.2: Components for Managed Forgetting – First Release*. February 2014

[ForgetIT 2014b] FORGETIT: *Deliverable D4.2: Information Analysis, Consolidation and Concentration Techniques, and Evaluation – First Release*. February 2014

[ForgetIT 2014c] FORGETIT: *Deliverable D5.2: Workflow Model and Prototype for Transition between Active System and AIS*. February 2014

[ForgetIT 2014d] FORGETIT: *Deliverable D6.2: Contextualisation Tools – First Release*. February 2014

[ForgetIT 2014e] FORGETIT: *Deliverable D7.2: Computational Storage Services – First Release*. February 2014

[ForgetIT 2014f] FORGETIT: *Deliverable D8.2: Preserve-or-Forget Reference Model – Initial Model*. September 2014

[ForgetIT 2014g] FORGETIT: *Deliverable D9.2: Use Cases & Mock-up Development*. February 2014

[ForgetIT 2015a] FORGETIT: *Deliverable D8.4: Preserve-or-Forget Framework – Second Release*. April 2015

[ForgetIT 2015b] FORGETIT: *Deliverable D8.5: Preserve-or-Forget Reference Model – Final Model*. February 2015

[Henjes u. a. 2007] HENJES, Robert ; SCHLOSSER, Daniel ; MENTH, Michael ; HIMMLER, Valentin: Throughput performance of the ActiveMQ JMS server. In: *Kommunikation in Verteilten Systemen (KiVS)* Springer (Veranst.), 2007, S. 113–124

[Hohpe und Woolf 2003] HOHPE, Gregor ; WOOLF, Bobby: *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*. Boston, MA, USA : Addison-Wesley Longman Publishing Co., Inc., 2003. – ISBN 0321200683

[Rabinovici-Cohen u. a. 2008] RABINOVICI-COHEN, Simona ; FACTOR, ME ; NAOR, Dalit ; RAMATI, Leeat ; RESHEF, Petra ; RONEN, Shahar ; SATRAN, Julian ; GIARETTA, David L.: Preservation DataStores: New storage paradigm for preservation environments. In: *IBM Journal of Research and Development* 52 (2008), Nr. 4.5, S. 389–399

[Rabinovici-Cohen u. a. 2013] RABINOVICI-COHEN, Simona ; MARBERG, John ; NAGIN, Kenneth ; PEASE, David: PDS Cloud: Long term digital preservation in the cloud. In: *Cloud Engineering (IC2E), 2013 IEEE International Conference on* IEEE (Veranst.), 2013, S. 38–45

[Snyder u. a. 2011] SNYDER, Bruce ; BOSNANAC, Dejan ; DAVIES, Rob: *ActiveMQ in action*. Manning, 2011

A Demo of the first prototype

In this Section we describe a simple demo showing the integrated components and the implementation of the two priority workflows. The demo is presented as a sequence of steps, with the help of some application screenshots.

1. A preservation event is triggered by the `Active System` (see Figure 17): the way this event is generated is still under development, it could be a suggestion from a PoF Middleware component (`Forgettor`) or could be due to a scheduled activity.

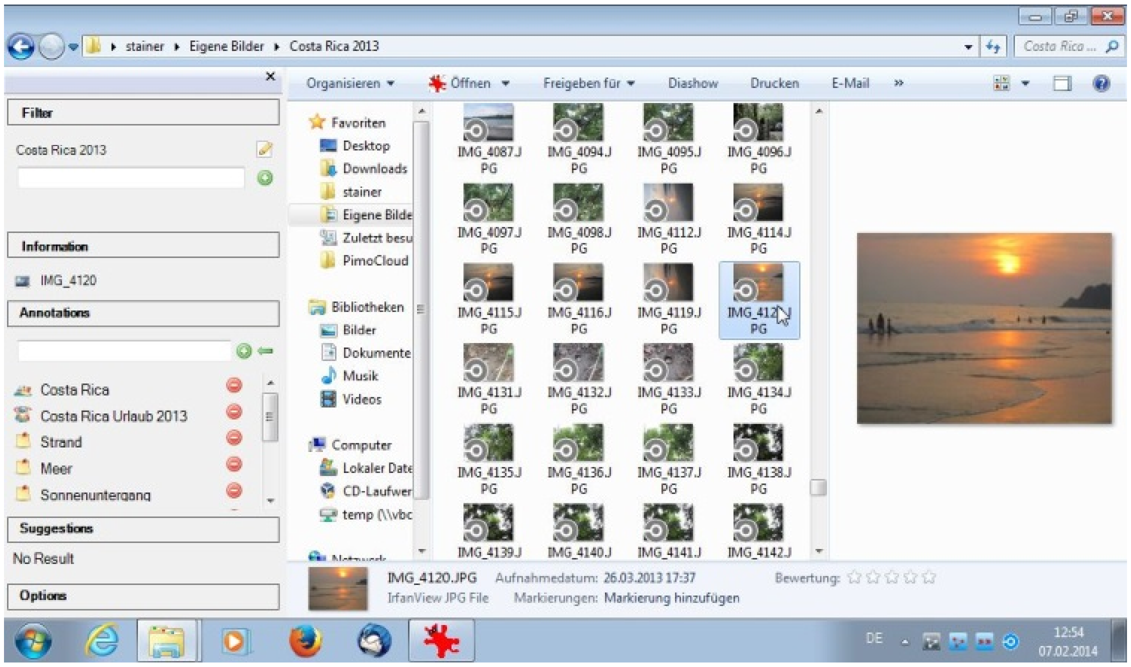


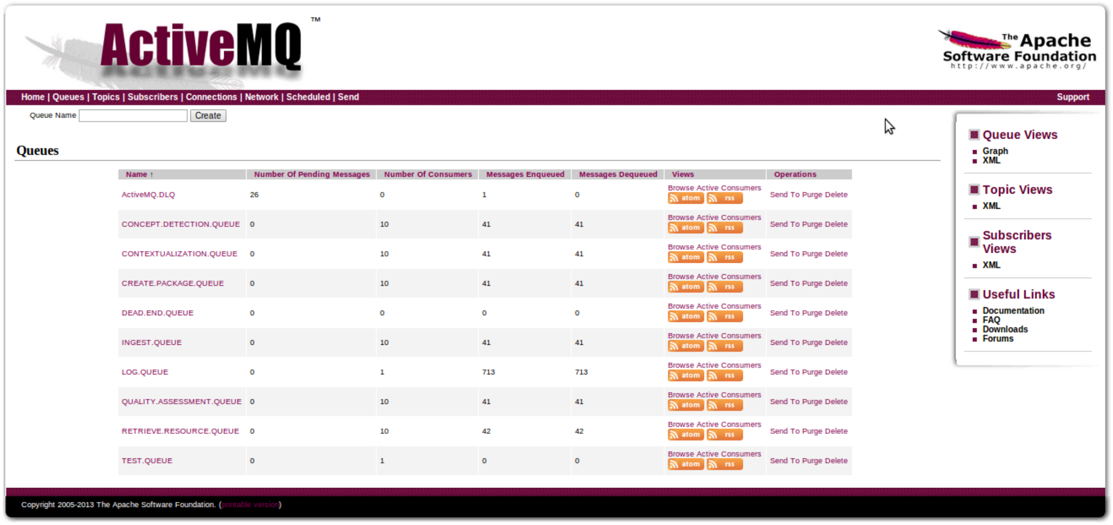
Figure 17: User interface of PIMO: selection of resource to be preserved.

2. A request is sent to the `PoF Middleware` via HTTP, using the REST APIs described in the previous Sections.

3. A Synergetic Preservation Worklow is triggered by the `WorkflowManager`, which creates a new job parsed by the `Scheduler`, which in turn sends the initial message to a queue on the messaging system (RETRIEVE.RESOURCE.QUEUE), as depicted in Figure 18, where the different queues available on the broker are shown, as well as the running instances of message processors (Consumers) listening to each queue and the number of pending and consumed messages.

4. The resource is retrieved from the CMIS server published by the `Active System`: the `Collector`, listening to RETRIEVE.RESOURCE.QUEUE, processes the message and uses the information therein (the resource URI) to retrieve the resource from the CMIS server.

5. The `ID Manager` generates a new ID to uniquely identify the content package.


ActiveMQ™
The Apache Software Foundation
http://www.apache.org/
Home | Queues | Topics | Subscribers | Connections | Network | Scheduled | Send
Support
Queue Name
Create
Queue Views
Graph
XML
Topic Views
XML
Subscribers Views
XML
Useful Links
Documentation
FAQ
Downloads
Forums
Copyright 2005-2013 The Apache Software Foundation. (printable version)

Queues

Name ↑	Number Of Pending Messages	Number Of Consumers	Messages Enqueued	Messages Dequeued	Views	Operations
ActiveMQ.DLQ	26	0	1	0	Browse Active Consumers atom rss	Send To Purge Delete
CONCEPT.DETECTION.QUEUE	0	10	41	41	Browse Active Consumers atom rss	Send To Purge Delete
CONTEXTUALIZATION.QUEUE	0	10	41	41	Browse Active Consumers atom rss	Send To Purge Delete
CREATE.PACKAGE.QUEUE	0	10	41	41	Browse Active Consumers atom rss	Send To Purge Delete
DEAD.END.QUEUE	0	0	0	0	Browse Active Consumers atom rss	Send To Purge Delete
INGEST.QUEUE	0	10	41	41	Browse Active Consumers atom rss	Send To Purge Delete
LOG.QUEUE	0	1	713	713	Browse Active Consumers atom rss	Send To Purge Delete
QUALITY.ASSESSMENT.QUEUE	0	10	41	41	Browse Active Consumers atom rss	Send To Purge Delete
RETRIEVE.RESOURCE.QUEUE	0	10	42	42	Browse Active Consumers atom rss	Send To Purge Delete
TEST.QUEUE	0	1	0	0	Browse Active Consumers atom rss	Send To Purge Delete

Figure 18: Queues published by the messaging system for the different workflow steps

6. The `Collector` stores the retrieved resource on the PoF Middleware server.

7. Flow control is returned to `Scheduler`, new messages containing resource information are sent to next queues in the workflow (QUALITY.ASSESSMENT.QUEUE, CONCEPT.DETECTION.QUEUE, CONTEXTUALIZATION.QUEUE) as shown in Figure 18, in order to trigger other components required for asynchronous and parallel content processing.

8. The `Extractor`, listening to the queues QUALITY.ASSESSMENT.QUEUE and CONCEPT.DETECTION.QUEUE, processes all messages and retrieves information required for image analysis (QA and concept detection), such as the resource path on the server and the content ID. Examples of results from preliminary QA and concept detection are shown below.

```
<?xml version="1.0" encoding="ISO-8859-15" standalone="no"?><QualityMeasure_detection>
    <QualityMeasures_list>
        <QualityMeasure id="C1">Blur</QualityMeasure>
        <QualityMeasure id="C2">Contrast</QualityMeasure>
        <QualityMeasure id="C3">Darkness</QualityMeasure>
        <QualityMeasure id="C4">Noise</QualityMeasure>
    </QualityMeasures_list>
    <QualityMeasures_order>C1 C2 C3 C4</QualityMeasures_order>
    <Image_QualityMeasures_List>
        <Image user_id="user_522\1001IMG_117.jpg">
            <image_url>
                http://middleware/pubstore/7cfc1a06-cbd7-4ba4-8c7e-e0d611b436e9/IMG_117.jpg
            </image_url>
            <image_id>1</image_id>
            <confidence_scores>0.682564 0.128376 0.29313 0.106106</confidence_scores>
            <imagequality_score>0.63686</imagequality_score>
        </Image>
    </Image_QualityMeasures_List>
</QualityMeasure_detection>
```

```
<?xml version="1.0" encoding="ISO-8859-15" standalone="no"?><Concept_detection>
    <Concepts_list>
      <concept id="C1">3_Or_More_People</concept>
      <concept id="C2">Actor</concept>
      ....
      <concept id="C157">Windows</concept>
    </Concepts_list>
    <Concepts_order>C1 C2 C3 C4 C5 C6 C7 C8 C9 C10 C11 C12 C13 C14 C15 C16 C17 C18 C19 C20
    C21 C22 C23 C24 C25 C26 C27 C28 C29 C30 C31 C32 C33 C34 C35 C36 C37 C38 C39 C40 C41 C42
    C43 C44 C45 C46 C47 C48 C49 C50 C51 C52 C53 C54 C55 C56 C57 C58 C59 C60 C61 C62 C63 C64
    C65 C66 C67 C68 C69 C70 C71 C72 C73 C74 C75 C76 C77 C78 C79 C80 C81 C82 C83 C84 C85 C86
    C87 C88 C89 C90 C91 C92 C93 C94 C95 C96 C97 C98 C99 C100 C101 C102 C103 C104 C105 C106
    C107 C108 C109 C110 C111 C112 C113 C114 C115 C116 C117 C118 C119 C120 C121 C122 C123
    C124 C125 C126 C127 C128 C129 C130 C131 C132 C133 C134 C135 C136 C137 C138 C139 C140
    C141 C142 C143 C144 C145 C146 C147 C148 C149 C150 C151 C152 C153 C154 C155 C156 C157
    </Concepts_order>
    <Image_Concepts_List>
        <Image user_id="C:\apache-tomcat-6.0.9\webapps\ROOT\CERTH_BIN\
            ForgetIT_Concept_Detection_Service\Images\user_594\1001IMG_3035.JPG">
            <image_url>http://middleware/pubstore/d78d2bdd-49b7-45c2-ab3c-22e3ff520d87/
                IMG_3035.JPG</image_url>
            <image_id>1</image_id>
            <confidence_scores>
                0.0039090286666667
                0.05104722
                ...
                0.449944026666667
            </confidence_scores>
        </Image>
    </Image_Concepts_List>
</Concept_detection>
```

9. The `Contextualizer` processes messages for text contextualization (mentions and context) retrieved from CONTEXTUALIZATION.QUEUE. In the following a short excerpt of a text document from TYPO3 CMS is shown, with the extracted mentions. The context ntuples are also created (being a huge file, it is not shown here).

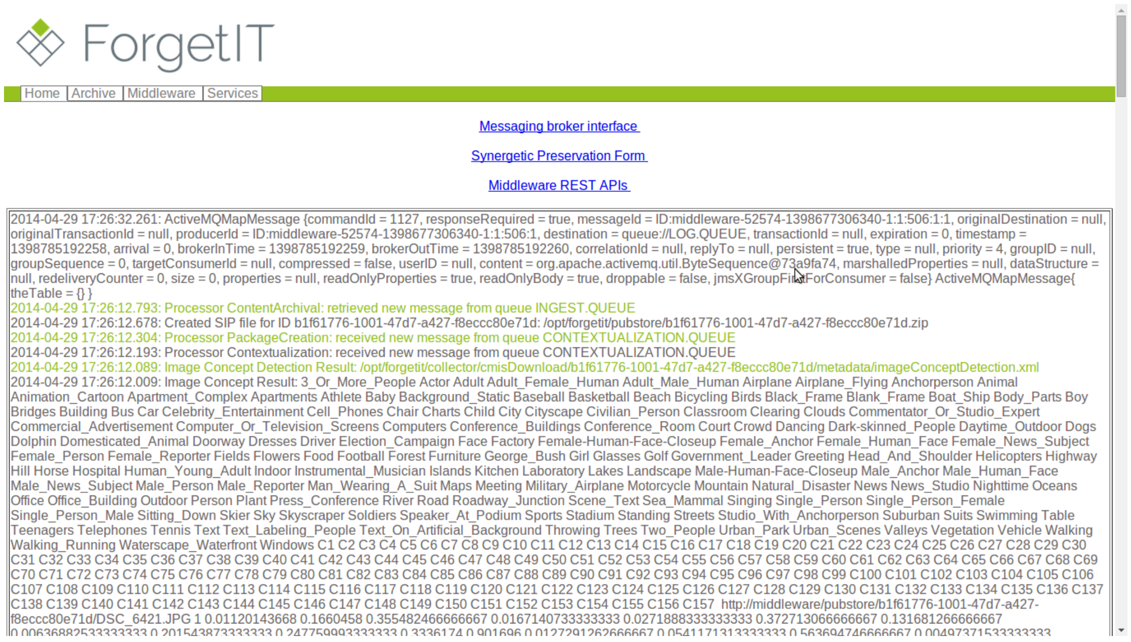
```
TYPO3 celebrates it's 20th anniversary. Having been one of the very first Content Management
Systems on the market 20 years of market leadership is an achievement truly worth
celebrating. Let's take a minute to reflect on the 20 years passed:
<ul><li>1997: The idea for TYPO3 is born</li><li>2000: The TYPO3 community is born<li>
<li>2005: the first TYPO3 conference is held</li><li>2008: TYPO3 reaches a market share of
close to 40\% in central Europe</li><li>2010: TYPO3 5.0 shakes up the CMS world</li>
<li>2014: 95\% of the world's websites run on Open Source and 60\% on TYPO3</li><li>2017:
Happy 20th Birthday TYPO3</li></ul> All the TYPO3 enthusiasts in the world:
Keep up the great work!
```

```
http://dbpedia.org/resource/Open_source
http://dbpedia.org/resource/Content_management_system
http://dbpedia.org/resource/Happiness
http://dbpedia.org/resource/TYPO3
http://dbpedia.org/resource/Europe
http://dbpedia.org/resource/Birthday
```

This process is executed in parallel to image analysis and, with the current implementation, it is only applied to text documents. It is worth mention that the last two steps are executed asynchronously, as shown in Figure 19.

10. Flow control is returned to `Scheduler`, messages containing processing results are sent to the queue used for SIP preparation (CREATE.PACKAGE.QUEUE).

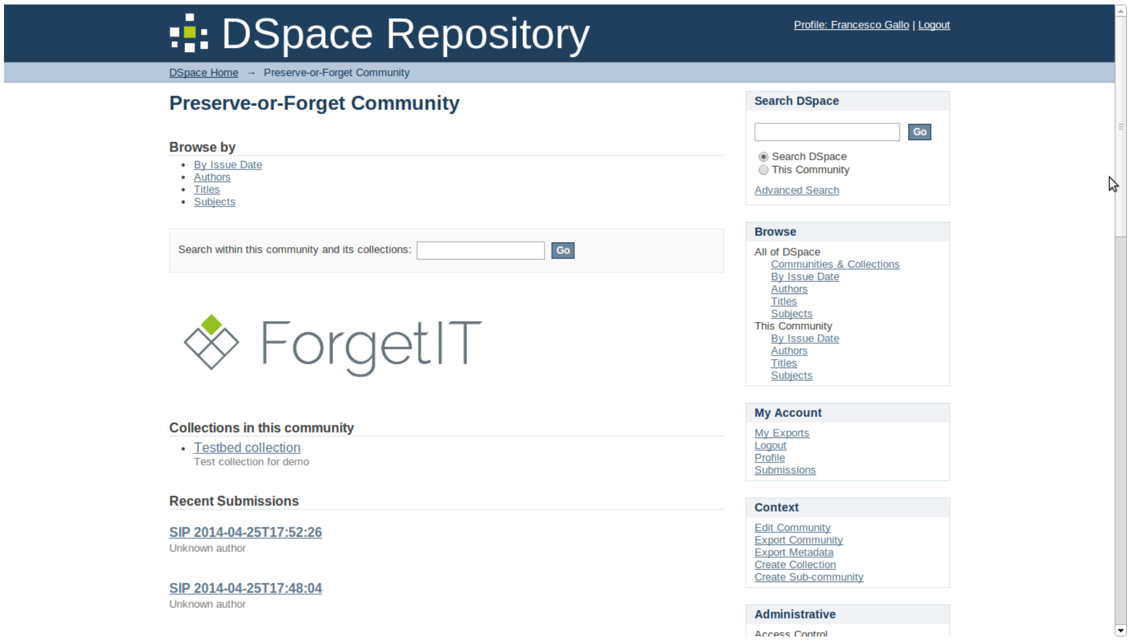
11. The `Archiver`, listening to CREATE.PACKAGE.QUEUE, processes messages with requests of SIP preparation, retrieving all resources and metadata. When the SIP is



```
2014-04-29 17:26:32.261: ActiveMQMapMessage {commandId = 1127, responseRequired = true, messageId = ID:middleware-52574-1398677306340-1:1:506:1:1, originalDestination = null,
originalTransactionId = null, producerId = ID:middleware-52574-1398677306340-1:1:506:1, destination = queue://LOG.QUEUE, transactionId = null, expiration = 0, timestamp =
1398785192258, arrival = 0, brokerInTime = 1398785192259, brokerOutTime = 1398785192260, correlationId = null, replyTo = null, persistent = true, type = null, priority = 4, groupID = null,
groupSequence = 0, targetConsumerId = null, compressed = false, userID = null, content = org.apache.activemq.util.ByteSequence@73a9fa74, marshalledProperties = null, dataStructure =
null, redeliveryCounter = 0, size = 0, properties = null, readOnlyProperties = true, readOnlyBody = true, droppable = false, jmsXGroupFirstForConsumer = false} ActiveMQMapMessage{
theTable = {} }
2014-04-29 17:26:12.793: Processor ContentArchival: retrieved new message from queue INGEST.QUEUE
2014-04-29 17:26:12.678: Created SIP file for ID b1f61776-1001-47d7-a427-f8eccc80e71d: /opt/forgetit/pubstore/b1f61776-1001-47d7-a427-f8eccc80e71d.zip
2014-04-29 17:26:12.304: Processor PackageCreation: received new message from queue CONTEXTUALIZATION.QUEUE
2014-04-29 17:26:12.193: Processor Contextualization: received new message from queue CONTEXTUALIZATION.QUEUE
2014-04-29 17:26:12.089: Image Concept Detection Result: /opt/forgetit/collector/cmisDownload/b1f61776-1001-47d7-a427-f8eccc80e71d/metadata/imageConceptDetection.xml
2014-04-29 17:26:12.009: Image Concept Result: 3_Or_More_People Actor Adult Adult_Female_Human Adult_Male_Human Airplane Airplane_Flying Anchorperson Animal
Animation_Cartoon Apartment_Complex Apartments Athlete Baby Background_Static Baseball Basketball Beach Bicycling Birds Black_Frame Blank_Frame Boat_Ship Body_Parts Boy
Bridges Building Bus Car Celebrity_Entertainment Cell_Phones Chair Charts Child City Cityscape Civilian_Person Classroom Clearing Clouds Commentator_Or_Studio_Expert
Commercial_Advertisement Computer_Or_Television_Screens Computers Conference_Buildings Conference_Room Court Crowd Dancing Dark-skinned_People Daytime_Outdoor Dogs
Dolphin Domesticated_Animal Doorway Dresses Driver Election_Campaign Face Factory Female-Human-Face-Closeup Female_Anchor Female_Human_Face Female_News_Subject
Female_Person Female_Reporter Fields Flowers Food Football Forest Furniture George_Bush Girl Glasses Golf Government_Leader Greeting Head_And_Shoulder Helicopters Highway
Hill Horse Hospital Human_Young_Adult Indoor Instrumental_Musician Islands Kitchen Laboratory Lakes Landscape Male-Human-Face-Closeup Male_Anchor Male_Human_Face
Male_News_Subject Male_Person Male_Reporter Man_Wearing_A_Suit Maps Meeting Military_Airplane Motorcycle Mountain Natural_Disaster News News_Studio Nighttime Oceans
Office Office_Building Outdoor Person Plant Press_Conference River Road Roadway_Junction Scene_Text Sea_Mammal Singing Single_Person Single_Person_Female
Single_Person_Male Sitting_Down Skier Sky Skyscraper Soldiers Speaker_At_Podium Sports Stadium Standing Streets Studio_With_Anchorperson Suburban Suits Swimming Table
Teenagers Telephones Tennis Text Text_Labeling_People Text_On_Artificial_Background Throwing Trees Two_People Urban_Park Urban_Scenes Valleys Vegetation Vehicle Walking
Walking_Running Waterscape_Waterfront Windows C1 C2 C3 C4 C5 C6 C7 C8 C9 C10 C11 C12 C13 C14 C15 C16 C17 C18 C19 C20 C21 C22 C23 C24 C25 C26 C27 C28 C29 C30
C31 C32 C33 C34 C35 C36 C37 C38 C39 C40 C41 C42 C43 C44 C45 C46 C47 C48 C49 C50 C51 C52 C53 C54 C55 C56 C57 C58 C59 C60 C61 C62 C63 C64 C65 C66 C67 C68 C69
C70 C71 C72 C73 C74 C75 C76 C77 C78 C79 C80 C81 C82 C83 C84 C85 C86 C87 C88 C89 C90 C91 C92 C93 C94 C95 C96 C97 C98 C99 C100 C101 C102 C103 C104 C105 C106
C107 C108 C109 C110 C111 C112 C113 C114 C115 C116 C117 C118 C119 C120 C121 C122 C123 C124 C125 C126 C127 C128 C129 C130 C131 C132 C133 C134 C135 C136 C137
C138 C139 C140 C141 C142 C143 C144 C145 C146 C147 C148 C149 C150 C151 C152 C153 C154 C155 C156 C157  http://middleware/pubstore/b1f61776-1001-47d7-a427-
f8eccc80e71d/DSC_6421.JPG 1 0.01120143668 0.1660458 0.355482466666667 0.0167140733333333 0.0271888333333333 0.372713066666667 0.131681266666667
0.00636882533333333 0.201543873333333 0.247759993333333 0.3336174 0.901696 0.0127291262666667 0.0541171313333333 0.563694746666667 0.00497371533333333
```

Figure 19: PoF Middleware GUI: log messages for all asynchronous processes.

ready, it is ingested into the `Preservation System` using REST APIs described in previous Sections. REST APIs exposed by the `Preservation System` are used to send the SIP via HTTP POST request. DSpace is configured by the `Digital Repository` administrator, with a defined ForgetIT community and a testbed collection, as shown in Figure 20.



Preserve-or-Forget Community

Browse by
- By Issue Date
- Authors
- Titles
- Subjects

Search within this community and its collections: Go

Collections in this community
- Testbed collection
  Test collection for demo

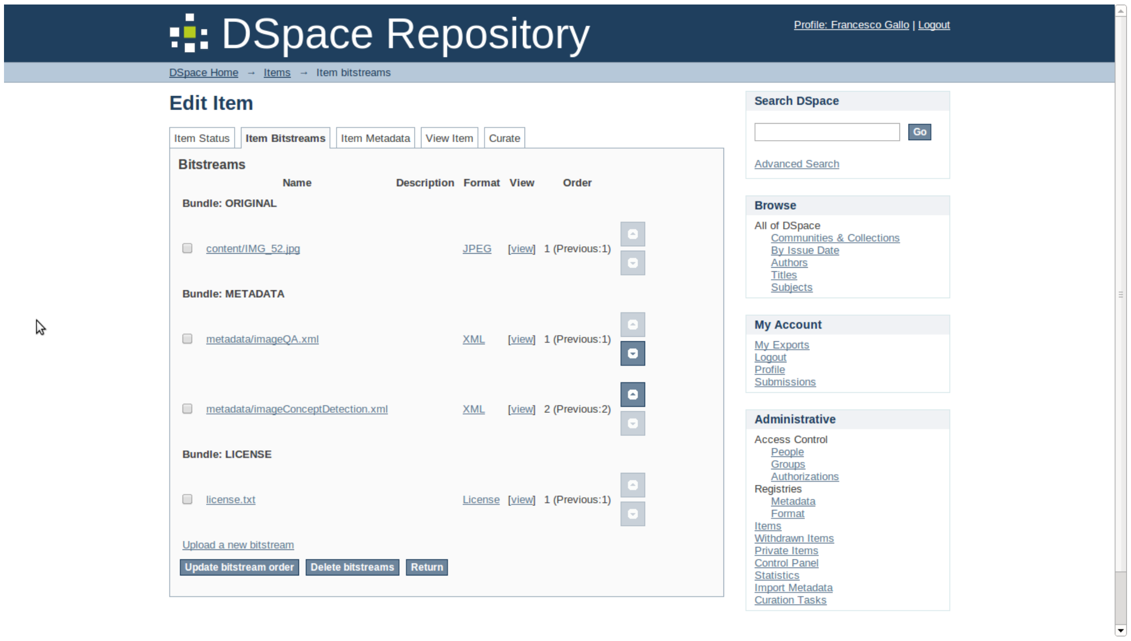
Recent Submissions

SIP 2014-04-25T17:52:26
Unknown author

SIP 2014-04-25T17:48:04
Unknown author

Figure 20: DSpace administrative GUI, with collections and communities.

12. The SIP structure contains a manifest file based on METS and two folders for content and metadata (see WP5 deliverables). This SIP is compliant to DSpace data model and can be accessed also using DSpace admin GUI (see Figure 21). We use the same structure also for the AIP, allowing two different export formats for the DIP, based on DSpace functionalities: one based on a METS profile and one defined as Simple Archive Format with Dublin Core manifest file (see Figure 22). An example of METS file associated is shown in the following.


DSpace Repository
Profile: Francesco Gallo | Logout
DSpace Home → Items → Item bitstreams

Edit Item

Bitstreams

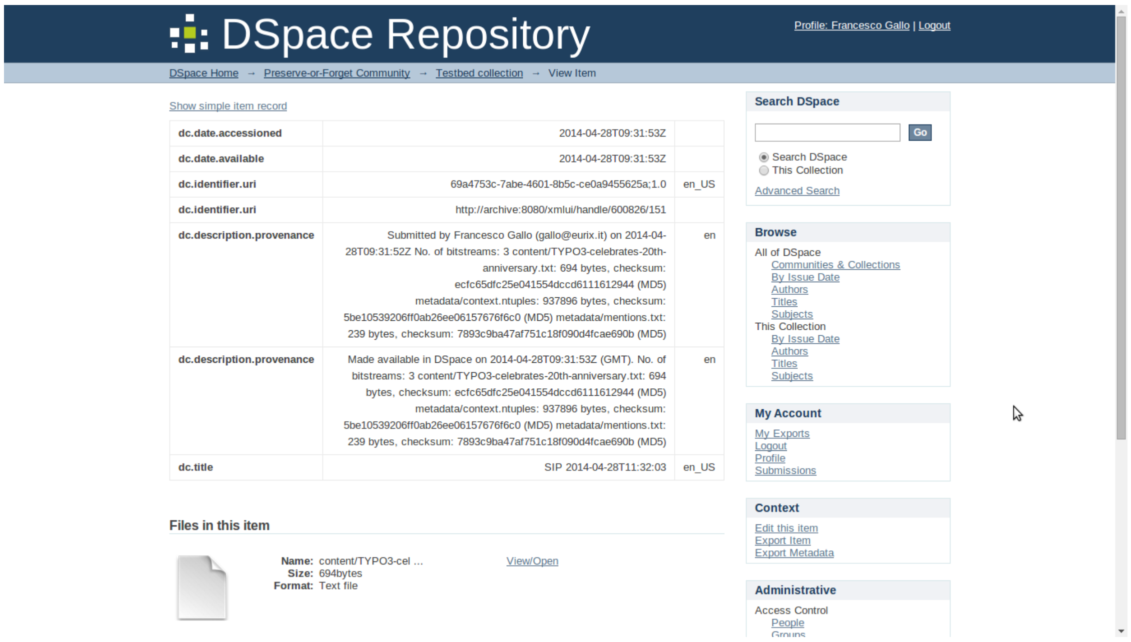
	Name	Description	Format	View	Order
Bundle: ORIGINAL					
☐	content/IMG_52.jpg		JPEG	[view]	1 (Previous:1)
Bundle: METADATA					
☐	metadata/imageQA.xml		XML	[view]	1 (Previous:1)
☐	metadata/imageConceptDetection.xml		XML	[view]	2 (Previous:2)
Bundle: LICENSE					
☐	license.txt		License	[view]	1 (Previous:1)

Figure 21: AIP preview: bitstreams and metadata files.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<mets xmlns="http://www.loc.gov/METS/" xmlns:ns2="http://www.w3.org/1999/xlink"
xmlns:ns3="http://www.loc.gov/mods/v3" ID="cbcff166-809d-46ac-a2bb-cf124df47365"
OBJID="d78d2bdd-49b7-45c2-ab3c-22e3ff520d87" TYPE="DSpace ITEM"
PROFILE="DSpace METS SIP Profile 1.0">
  <dmdSec ID="dmdSec_1">
    <mdWrap MDTYPE="MODS">
      <xmlData>
        <ns3:mods>
          <ns3:titleInfo>
            <ns3:title>SIP 2014-04-24T18:51:55</ns3:title>
          </ns3:titleInfo>
          <ns3:identifier type="uri">
            pimocloud:586a6de3-9aa6-4832-b6c4-f98e0beaaf41
          </ns3:identifier>
        </ns3:mods>
      </xmlData>
    </mdWrap>
  </dmdSec>
  <fileSec>
    <fileGrp USE="ORIGINAL">
      <file ID="cn-file-0001" MIMETYPE="image/jpeg" SIZE="1779948"
       CHECKSUM="3c00d41925ffd8507773ca979ef97ae6" CHECKSUMTYPE="MD5">
        <FLocat LOCTYPE="URL" ns2:type="simple" ns2:href="content/IMG_3035.JPG"/>
      </file>
    </fileGrp>
    <fileGrp USE="METADATA">
      <file ID="md-file-0002" MIMETYPE="application/xml" SIZE="906"
```

```
        CHECKSUM="50807131447d3718e724f837a8a4cd47" CHECKSUMTYPE="MD5">
          <FLocat LOCTYPE="URL" ns2:type="simple" ns2:href="metadata/imageQA.xml"/>
        </file>
       <file ID="md-file -0003" MIMETYPE="application/xml" SIZE="11174"
        CHECKSUM="1c6eea66af562726641fda016405076b" CHECKSUMTYPE="MD5">
          <FLocat LOCTYPE="URL" ns2:type="simple"
            ns2:href="metadata/imageConceptDetection.xml"/>
        </file>
      </fileGrp>
    </fileSec>
    <structMap>
      <div DMDID="dmdSec_1">
        <div LABEL="FILES">
          <fptr ID="fptr -0001" FILEID="cn-file -0001"/>
          <fptr ID="fptr -0002" FILEID="md-file -0002"/>
          <fptr ID="fptr -0003" FILEID="md-file -0003"/>
        </div>
      </div>
    </structMap>
</mets>
```

13. The `ID Manager` updates IDs associated to the content adding the AIP ID provided by the `Digital Repository` (see Figure 23). For each content the resource URI is mapped to the AIP ID and to other IDs, as explained in the following steps.



dc.date.accessioned	2014-04-28T09:31:53Z	
dc.date.available	2014-04-28T09:31:53Z	
dc.identifier.uri	69a4753c-7abe-4601-8b5c-ce0a9455625a;1.0	en_US
dc.identifier.uri	http://archive:8080/xmlui/handle/600826/151	
dc.description.provenance	Submitted by Francesco Gallo (gallo@eurix.it) on 2014-04-28T09:31:52Z No. of bitstreams: 3 content/TYPO3-celebrates-20th-anniversary.txt: 694 bytes, checksum: ecfc65dfc25e041554dccd6111612944 (MD5) metadata/context.ntuples: 937896 bytes, checksum: 5be10539206ff0ab26ee06157676f6c0 (MD5) metadata/mentions.txt: 239 bytes, checksum: 7893c9ba47af751c18f090d4fcae690b (MD5)	en
dc.description.provenance	Made available in DSpace on 2014-04-28T09:31:53Z (GMT). No. of bitstreams: 3 content/TYPO3-celebrates-20th-anniversary.txt: 694 bytes, checksum: ecfc65dfc25e041554dccd6111612944 (MD5) metadata/context.ntuples: 937896 bytes, checksum: 5be10539206ff0ab26ee06157676f6c0 (MD5) metadata/mentions.txt: 239 bytes, checksum: 7893c9ba47af751c18f090d4fcae690b (MD5)	en
dc.title	SIP 2014-04-28T11:32:03	en_US

Files in this item

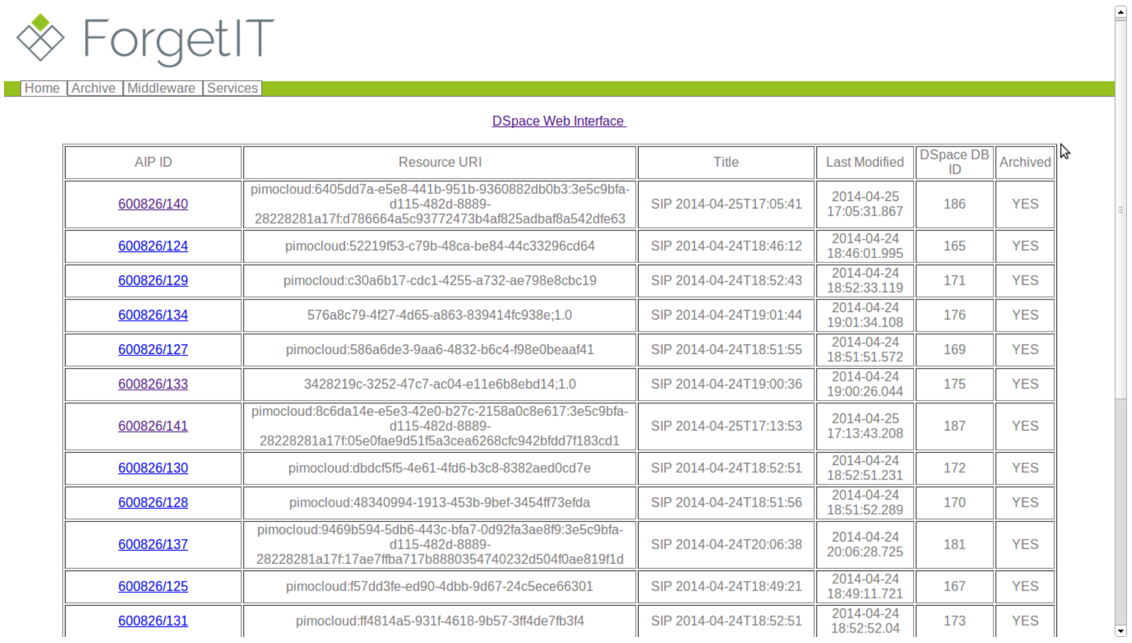
Name: content/TYPO3-cel ... View/Open
Size: 694bytes
Format: Text file

Figure 22: AIP preview: Dublin Core metadata, with identifier and provenance information.

14. The `Packager` component in the `Preservation System` stores the AIP into the `Cloud Storage Service` using PDS REST APIs.

15. The `ID Manager` updates IDs associated to the content adding PDS ID provided by the `Cloud Storage Service`.

16. The `Cloud Storage Service` triggers Storlets according to rules.



DSpace Web Interface

AIP ID	Resource URI	Title	Last Modified	DSpace DB ID	Archived
600826/140	pimocloud:6405dd7a-e5e8-441b-951b-9360882db0b3:3e5c9bfa-d115-482d-8889-28228281a17f:d786664a5c93772473b4af825adbaf8a542dfe63	SIP 2014-04-25T17:05:41	2014-04-25 17:05:31.867	186	YES
600826/124	pimocloud:52219f53-c79b-48ca-be84-44c33296cd64	SIP 2014-04-24T18:46:12	2014-04-24 18:46:01.995	165	YES
600826/129	pimocloud:c30a6b17-cdc1-4255-a732-ae798e8cbc19	SIP 2014-04-24T18:52:43	2014-04-24 18:52:33.119	171	YES
600826/134	576a8c79-4f27-4d65-a863-839414fc938e;1.0	SIP 2014-04-24T19:01:44	2014-04-24 19:01:34.108	176	YES
600826/127	pimocloud:586a6de3-9aa6-4832-b6c4-f98e0beaaf41	SIP 2014-04-24T18:51:55	2014-04-24 18:51:51.572	169	YES
600826/133	3428219c-3252-47c7-ac04-e11e6b8ebd14;1.0	SIP 2014-04-24T19:00:36	2014-04-24 19:00:26.044	175	YES
600826/141	pimocloud:8c6da14e-e5e3-42e0-b27c-2158a0c8e617:3e5c9bfa-d115-482d-8889-28228281a17f:05e0fae9d51f5a3cea6268cfc942bfdd7f183cd1	SIP 2014-04-25T17:13:53	2014-04-25 17:13:43.208	187	YES
600826/130	pimocloud:dbdcf5f5-4e61-4fd6-b3c8-8382aed0cd7e	SIP 2014-04-24T18:52:51	2014-04-24 18:52:51.231	172	YES
600826/128	pimocloud:48340994-1913-453b-9bef-3454ff73efda	SIP 2014-04-24T18:51:56	2014-04-24 18:51:52.289	170	YES
600826/137	pimocloud:9469b594-5db6-443c-bfa7-0d92fa3ae8f9:3e5c9bfa-d115-482d-8889-28228281a17f:17ae7ffba717b8880354740232d504f0ae819f1d	SIP 2014-04-24T20:06:38	2014-04-24 20:06:28.725	181	YES
600826/125	pimocloud:f57dd3fe-ed90-4dbb-9d67-24c5ece66301	SIP 2014-04-24T18:49:21	2014-04-24 18:49:11.721	167	YES
600826/131	pimocloud:ff4814a5-931f-4618-9b57-3ff4de7fb3f4	SIP 2014-04-24T18:52:51	2014-04-24 18:52:52.04	173	YES

Figure 23: Web interface of the PoF Middleware, the different IDs associated to the same content are shown, as well as the preservation status.

17. The status of the resource is updated: resource is shown as *preserved* in the `Active System`.

After the content has been successfully preserved, a workflow to restore one or more resources can be triggered, as described below. This workflow can be triggered for different reasons, e.g. in order to replace a corrupted or modified resource with the original one stored in the `Preservation System`.

1. The `Active System` can send a request to `PoF Middleware` using the REST APIs, in order to restore or update a resource locally.
2. A Resource Restore Worklow is triggered by the `WorkflowManager`, which creates a new job, parsed by the `Scheduler`, which in turn creates the initial message and sends it to the specific queue available on the messaging system.
3. The `Collector`, listening to a specific queue, retrieves the message and extracts the relevant information (resource URI).
4. The `ID Manager` returns the AIP ID associated to the resource URI.
5. The `Collector` retrieves the DIP from the `Preservation System`, using the REST APIs, The DIP contains the original resource (or a new one after transformation by PDS), which is returned to the `Active System`.
6. The user is notified after the resource is updated.

The workflows described above will be used as starting point for future more complex workflows which will be implemented in the next releases of the framework.

B DSpace Installation Guide

B.1 Introduction

Information about DSpace can be found on the project web site[21]. The role of DSpace in the PoF Framework and the integration with the other components in the overall architecture is described in deliverable D8.1 [ForgetIT 2013d]. Additional information can be found in D7.2 [ForgetIT 2014e] (integration with cloud storage) and in D5.2 [ForgetIT 2014c] (synergetic preservation workflows). This guide is based on official DSpace documentation[22], tailored to Ubuntu Server 12.04 LTS, with additional configuration information for the PoF Framework. Other applications and libraries required to install and run DSpace are Apache Maven[23] and Apache Ant[24] for compiling and building DSpace sources, PostgreSQL[25] as internal DB used by DSpace and Apache Tomcat[26] for the runtime.

B.2 Installation procedure

The following instructions have been tested with DSpace 4.1. If you are using a different version of DSpace, you should check the documentation available on DSpace web site. In order to install DSpace using the following instructions, you need a basic installation of Ubuntu Server 12.04 LTS. Please refer to Ubuntu documentation for the installation of the operating system. You can use a physical or virtual machine for installing Ubuntu, as done for example in the ForgetIT testbed. In the following we assume that you have installed Ubuntu and that you have access to the machine using either the `root` user or any user belonging to the `sudo` group. During the Ubuntu installation, it is advisable to include an OpenSSH server as additional software, mainly if you are installing DSpace in a virtual machine hosted by a remote server. Please note that in the instructions below, after the creation of the DSpace user, you need to start DSpace and apply any modifications to the DSpace configuration using this user only, who must also have writing permissions for all the directories used by DSpace.

Configuration of Ubuntu

From within a terminal, update the Ubuntu installation and reboot the machine:

```
$ sudo apt-get update
$ sudo apt-get upgrade
$ sudo reboot now
```

[21]DSpace - `http://www.dspace.org`
[22]DSpace Guide - `https://wiki.duraspace.org/display/DSDOC/All+Documentation`
[23]Apache Maven - `http://maven.apache.org`
[24]Apache Ant - - `http://ant.apache.org`
[25]PostgreSQL - `http://www.postgresql.org`
[26]Apache Tomcat - `http://tomcat.apache.org`

Install the Java JDK 7[27] and Apache Maven with the following command:

```
  $ sudo apt-get install openjdk-7-jdk maven
```

Check the Maven installation running:

```
  $ mvn -version
```

You should get an output similar to the following:

```
Apache Maven 3.0.4
Maven home: /usr/share/maven
Java version: 1.7.0_51, vendor: Oracle Corporation
Java home: /usr/lib/jvm/java-7-openjdk-amd64
Default locale: it_IT, platform encoding: UTF-8
OS name: "linux", version: "3.8.0-35-generic", arch: "amd64",
family: "unix"
```

Since Maven on Ubuntu server will add version 6 of Java Runtime Environment as additional dependency, configure JRE in order to use version 7 (for Java compiler and other Java related utilities the used version should already be 7):

```
  $ sudo update-alternatives --config java
```

and select version 7 when prompted (option 2 in the example below):

```
There are 2 choices for the alternative java (providing /usr/bin/java).

Selection    Path                                            Priority   Status
------------------------------------------------------------------------------
* 0          /usr/lib/jvm/java-6-openjdk-amd64/jre/bin/java   1061      auto mode
  1          /usr/lib/jvm/java-6-openjdk-amd64/jre/bin/java   1061      manual mode
  2          /usr/lib/jvm/java-7-openjdk-amd64/jre/bin/java   1051      manual mode

Press enter to keep the current choice[*], or type selection number: 2
```

Install Apache Web Server, including the proxy module (Apache WS will act as a proxy to forward the requests to the DSpace web applications running on Tomcat, while static content, such URLs for published AIP files, will be managed by Apache WS itself):

```
  $ sudo apt-get install apache2 libapache2-mod-proxy-html
  libxml2-dev
```

Install libraries for file compression (zip, p7zip-full, p7zip-rar): some of the previous libraries are not mandatory, but can be useful for testing DSpace locally, e.g. creating zip files for the SIP or to open exported AIPs.

Install tools for CIFS, to mount storage folder on external devices (e.g. NAS). NFS or iSCSI could be used, too.

```
  $ sudo apt-get install cifs-tools
```

The automatic creation of new mount points must be added to `/etc/fstab` file, depending on your configuration.

[27]Open JDK - `http://openjdk.java.net`

Create “dspace” user

Create a new Linux user named `dspace` (with password `dspace`) and add it to the sudoers group with the following commands:

```
$ sudo adduser dspace
$ sudo adduser dspace sudo
```

From now on, switch to DSpace user `dspace`: you can either logout and then login again with user `dspace` or simply use the following command as `root`:

```
$ su dspace
```

Install and configure PostgreSQL 9.1

DSpace currently supports PostgreSQL and Oracle DB. PostgreSQL is the default choice and no additional drivers or adapters have to be added. Instructions below refer to the default installation with PostgreSQL. If you want to use Oracle DB please refer to the documentation available in the official DSpace documentation.

Install PostgreSQL and check the installation with the following commands:

```
$ sudo apt-get install postgresql-9.1
$ psql -- version
```

Edit PostgreSQL configuration files as described below:

- ***postgresql.conf*** in */etc/prostgresql/9.1/main/*: uncomment the line containing: *listen_addresses = ’localhost’* and edit the line to listen on all addresses, *listen_addresses = ’*’*

- ***pg_hba.conf*** in */etc/prostgresql/9.1/main/*: add the lines *host dspace dspace 127.0.0.1 255.255.255.255 md5* and *host all all 127.0.0.1/24 trust*.

Restart PostgreSQL:

```
$ sudo service postgresql restart
```

Switch to user `root` and then switch to user `postgres`:

```
$ sudo su -
$ su postgres
```

Create a new user for PostgreSQL, with username `dspace` and password `dspace` (this user is different from the one created on Ubuntu and can be changed in the DSpace configuration):

```
$ createuser -U postgres -d -A -P dspace
```

Create a new PostgreSQL DB schema, named `dspace`, owned by the `dspace` PostgreSQL user created above:

```
$ createdb --owner=dspace --encoding=UNICODE dspace
```

Switch back to Ubuntu user `dspace` issuing twice the shell command `exit`.

Create a folder for copying source files of the required applications and for setup. The suggested configuration is to create a folder under `/opt` and to assign ownership to `dspace` user, as shown below:

```
$ sudo mkdir /opt/forgetit
$ sudo chown -R dspace:dspace /opt/forgetit
$ mkdir /opt/forgetit/applications
$ mkdir /opt/forgetit/setup
```

Install Apache Tomcat 7

Install Apache Tomcat 7 as user `dspace` created in the previous section. Download Tomcat 7, e.g. using the following command (check the link for the current version):

```
$ cd /opt/forgetit/applications
$ wget -O apache-tomcat-7.0.52.tar.gz
  http://apache.panu.it/tomcat/tomcat-7/v7.0.52/bin/apache-tomcat
  -7.0.52.tar.gz
```

or copy the .tar.gz file downloaded with another machine using `scp` command (you need an OpenSSH server running, see instructions above).

Uncompress the Tomcat tar.gz file into directory `/opt/forgetit/tomcat7` (in the following this directory will be referred to as `[Tomcat_Install_Dir]`) :

```
$ tar -xzvf apache-tomcat-7.0.52.tar.gz -C /opt/forgetit
$ mv apache-tomcat-7.0.52 tomcat7
```

Instructions to configure a service for Tomcat to start at boot are provided in the last section. Edit the Tomcat 7 configuration as described below:

- create file ***setenv.sh*** in *[Tomcat_Install_Dir]/bin* and set *JAVA_OPTS="-Xmx512M -Xms64M -XX:MaxPermSize=256M -Dfile.encoding=UTF-8"*
- Edit file ***server.xml*** in *[Tomcat_Install_Dir]/conf* adding a configuration option to the *Connector* element: *URIEncoding="UTF-8"*

Install DSpace 4.1

Download DSpace 4.1 from DSpace web site, e.g. using the following command (check the link for the current version):

```
$ cd /opt/forgetit/applications
$ wget -O dspace-4.1-src-release.tar.gz  http://sourceforge.net/
      projects/dspace/files/DSpace\%20Stable/4.1/dspace-4.1-src-
      release.tar.gz/download
```

As `dspace` user unpack the DSpace tar.gz file. The extracted folder (e.g. `dspace-4.1-src-release`) will be referenced to as `[dspace-source]` in the following. Create a directory to install DSpace, e.g. `/opt/forgetit/dspace-4.1`, referred to as (`[dspace-install]`):

```
$ cd /opt/forgetit/applications
$ tar -xzvf dspace-4.1-src-release.tar.gz
$ mkdir /opt/forgetit/\emph{[dspace-install]}
$ cd /opt/forgetit/applications/\emph{[dspace-source]
```

Configure file ***build.properties***, editing the following properties:

- `dspace.install.dir` pointing to `[dspace-install]`
- `dspace.hostname` = `[HOSTNAME]` (set it to the hostname chosen during the installation, e.g. archive. Compare with hostname in file `/etc/hosts` or with environment variable `HOSTNAME`)
- `dspace.baseUrl` = http://`[HOSTNAME]`:8080
- `dspace.name` = DSpace for Preserve-or-Forget Framework (or any other name which is meaningful for you)
- `mail.server` = `YOUR_MAIL_SERVER`
- `mail.from.address` = `dspace-admin@forgetit-project.eu` (or change according to your configuration)
- `mail.feedback.recipient` = `CONTACT_USER_EMAIL`
- `mail.admin` = `DSPACE_ADMIN_EMAIL`
- uncomment the line `handle.canonical.prefix` = `$dspace.url/handle/` and comment the line `handle.canonical.prefix = http://hdl.handle.net/`, unless you want to subscribe to the handle service by CNRI
- `handle.prefix` = `ANY_VALUE` (use official prefix from handle service if available)

All properties above but the installation directory can be modified later, editing file `dspace.cfg`

Compile using Maven and install using Ant, with the following commands:

```
$ cd [dspace-source]
$ mvn package
$ cd [dspace-source]/dspace/target/dspace-[version]-build
$ ant fresh_install
```

DSpace is installed in the specified directory `[dspace-install]`.

Create an admin account for DSpace:

```
$ cd [dspace-install]
$ ./bin/dspace create-administrator
```

Deploy the created web applications (copy `[dspace-install]/webapps` content or create symlinks for all DSpace web applications in Tomcat webapps folder).

Start Tomcat 7 as `dspace` user (see Tomcat documentation for starting and stopping the server) and check the DSpace installation. Verify that DSpace is up and running at the following URLs (change `archive.forgetit-project.eu` with correct host):

- check DB connection with command: `$ ./bin/dspace test-database`
- check email settings: `$ ./bin/dspace test-email`
- `http://archive.forgetit-project.eu:8080/jspui` (JSP-based interface)
- `http://archive.forgetit-project.eu:8080/xmlui` (XML-based interface)
- sign in with administrator account created above on DSpace web interface
- try to create collections, new items, etc. (see instructions for Getting Started on DSpace web site)

To customize the home page of the XMLUI interface, edit file `/opt/forgetit/dspace-4.1/config/news-xmlui.xml`.

Optional configuration for Apache Web Server and Apache Tomcat

Apache WS can be configured to act as a reverse proxy for Tomcat (requests to DSpace are proxied by Apache WS and viceversa). Static content (e.g. AIP files exported from DSpace) is served by Apache WS.

Check that the proxy module is installed (see section about Ubuntu environment configuration) and enabled. Use command `sudo a2enmod proxy_http` and restart with `sudo service apache2 restart`).

Add the following directives to file `/etc/apache2/sites-available/default`, for each one of the applications in webapps to be proxied:

```
        ProxyPass /xmlui http://archive:8080/xmlui
        ProxyPassReverse /xmlui http://archive:8080/xmlui
```

According to the example above, the new URL of DSpace XML interface will be `http://archive/xmlui`.

Tomcat 7 can be configured to start at boot. Paste the example script below to a text file `tomcat7` and copy it to `/etc/init.d`, then execute command:

```
$ update-rc.d <nomescript> defaults
```

The script must be executable (use `chmod` command: `$ sudo chmod ugo+rx tomcat7`). Note that in the provided example Tomcat is run as user `dspace`.

```
#! /bin/sh

### BEGIN INIT INFO
# Provides:             Tomcat7
# Required-Start:       $remote_fs $syslog
# Required-Stop:        $remote_fs $syslog
# Default-Start:        2 3 4 5
# Default-Stop:
# Short-Description:    Tomcat7
### END INIT INFO

set -e
. /lib/lsb/init-functions
TOMCAT_HOME=/opt/tomcat7
TOMCAT_USER=dspace

case "$1" in
  start)
        log_daemon_msg "Starting Tomcat7"
        su - $TOMCAT_USER -c "$TOMCAT_HOME/bin/startup.sh > /dev/null"
        log_end_msg 0
        ;;
  stop)
        log_daemon_msg "Stopping Tomcat7"
        su - $TOMCAT_USER -c "$TOMCAT_HOME/bin/shutdown.sh > /dev/null"
        log_end_msg 0
        ;;
  reload|force-reload)
        ;;
  restart)
        ;;
  *)
        log_action_msg "Usage: /etc/init.d/tomcat7 {start|stop|reload|force-
        reload|restart|try-restart|status}" || true
        exit 1
esac

exit 0
```

C Message Oriented Middleware

C.1 Apache ActiveMQ

A brief introduction

Apache ActiveMQ[28] is an open source message broker written in Java, including a full-fledged JMS client. It provides "Enterprise Features" which in this case means fostering the communication from more than one client or server. Supported clients include the obvious Java via JMS 1.1 as well as several other "cross language" clients. The communication is managed with features such as computer clustering and ability to use any database as a JMS persistence provider besides virtual memory, cache, and journal persistence. The following guide explains how to install and run ActiveMQ and how to correctly deploy it into Apache Tomcat 7. The following instructions have been tested on Mac OS X 10.9.2 and Ubuntu 12.04 LTS.

Download and installation

First download the `tar` archive corresponding to the desired version from ActiveMQ web site, extract it into a folder of your choice and then change the permission of the start-up script. You can move the tar file into your folder (`[activemq_ install_ dir]`) and run the following command from the terminal:

```
$ tar -xvzf apache-activemq-5.9.0-bin.tar.gz
$ cd [activemq_install_dir]/bin
$ chmod 755 activemq
```

Switch to folder `[activemq_ install_ dir]` and run ActiveMQ using the following commands:

```
$ cd [activemq_install_dir]/bin
$ ./activemq start
```

You can now go to `http://localhost:8161/admin` to check the status of ActiveMQ. If ActiveMQ is ok you should see a page similar to Figure 24.

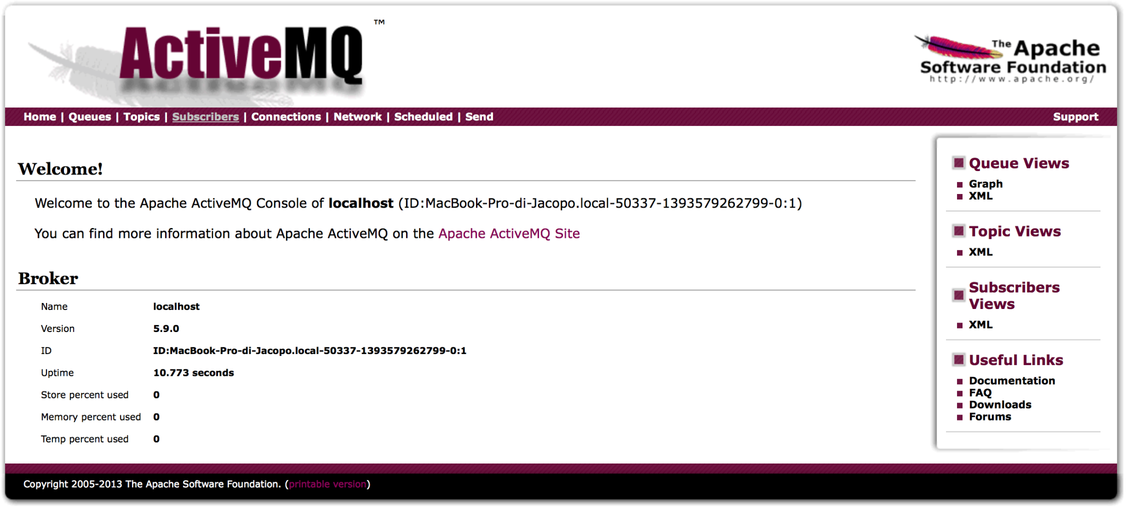
Run the following command from the terminal to stop ActiveMQ:

```
$ ./activemq stop
```

Deployment into Tomcat 7

After installing ActiveMQ, it is possible to download the web console and deploy it into the `webapps` folder of Tomcat 7. Choose an ActiveMQ version from URL `http://repo1.`

[28]Apache ActiveMQ - `http://activemq.apache.org`



Welcome!

Welcome to the Apache ActiveMQ Console of **localhost** (ID:MacBook-Pro-di-Jacopo.local-50337-1393579262799-0:1)

You can find more information about Apache ActiveMQ on the Apache ActiveMQ Site

Broker

Name	localhost
Version	5.9.0
ID	ID:MacBook-Pro-di-Jacopo.local-50337-1393579262799-0:1
Uptime	10.773 seconds
Store percent used	0
Memory percent used	0
Temp percent used	0

Copyright 2005-2013 The Apache Software Foundation. (printable version)

Figure 24: ActiveMQ web-console homepage

`maven.org/maven2/org/apache/activemq/activemq-web-console/` and download the corresponding `.war` file (for example: `activemq-web-console-5.9.0.war`). You should also download the appropriate `.jar` (for example: `activemq-all-5.9.0.jar`). Copy the `.war` file into `[TOMCAT_ HOME]/webapps` folder and the `.jar` file into `[TOMCAT_ HOME]/lib` folder.

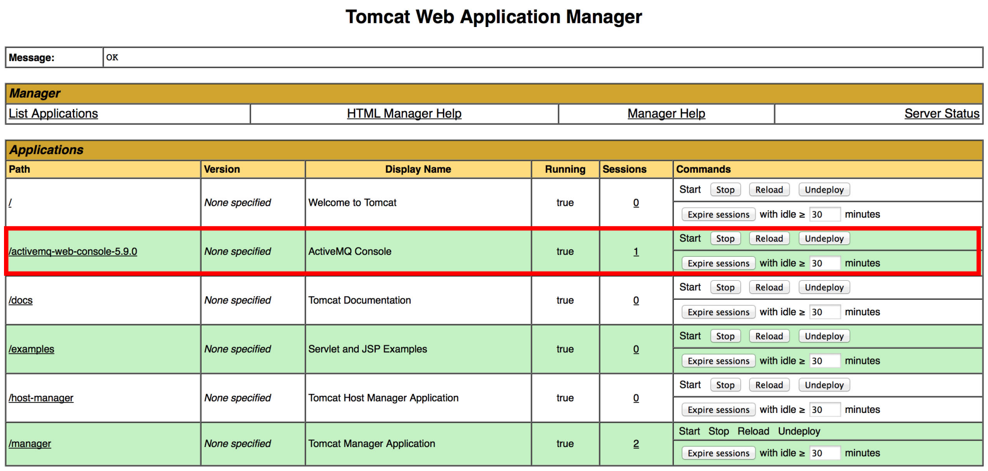
Now you can start Tomcat 7 with the following command and wait for the correct deployment of the web applications:

```
$ [TOMCAT_HOME]/bin/startup.sh
```

If the ActiveMQ web console was successfully deployed, you should see a message similar to the following:

```
Informazioni: Deploying web application archive /usr/local
/apache-tomcat-7.0.50/webapps/activemq-web-console-5.9.0.war
...
PListStore:[/usr/local/apache-tomcat-7.0.50/activemq-data/
web-console/tmp_storage] started
Using Persistence Adapter: KahaDBPersistenceAdapter[/usr/
local/apache-tomcat-7.0.50/${activemq.data}/kahadb]
JMX consoles can connect to service:jmx:rmi:///jndi/rmi://
localhost:1099/jmxrmi
Apache ActiveMQ 5.9.0 (web-console, ID:MacBook-Pro-di-Jacopo.
local-49790-1392715784927-0:1) is starting
Listening for connections at: tcp://localhost:61616
Connector openwire started
Listening for connections at: stomp://localhost:61613
Connector stomp started
Apache ActiveMQ 5.9.0 (web-console, ID:MacBook-Pro-di-Jacopo.
local-49790-1392715784927-0:1) started
For help or more information please see: http://activemq.
apache.org
...
```

Finally open the ActiveMQ web console from the Tomcat manager as shown in Figure 25.



Tomcat Web Application Manager

Message:	OK

Manager			
List Applications	HTML Manager Help	Manager Help	Server Status

Applications					
Path	Version	Display Name	Running	Sessions	Commands
/	*None specified*	Welcome to Tomcat	true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes
/activemq-web-console-5.9.0	*None specified*	ActiveMQ Console	true	1	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes
/docs	*None specified*	Tomcat Documentation	true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes
/examples	*None specified*	Servlet and JSP Examples	true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes
/host-manager	*None specified*	Tomcat Host Manager Application	true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes
/manager	*None specified*	Tomcat Manager Application	true	2	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes

Figure 25: Tomcat 7 manager

C.2 Apache Camel

Apache Camel[29] is a rule-based routing and mediation engine that provides a Java object-based implementation of the Enterprise Integration Patterns (EIP) using an API (or declarative Java Domain Specific Language) to configure routing and mediation rules. The main role of Apache Camel in the PoF Framework implementation is to provide a flexible rule engine to implement workflows executed within the PoF Middleware, leveraging the messaging system provided by Apache ActiveMQ.

Apache Camel has not been included yet in the first prototype implementation, but it is mentioned here because it is part of the same Apache ServiceMix[30] suite and will be used for the second PoF Framework release, where additional and more complex workflows will be defined. The main advantage of using Apache Camel here is that it can be integrated seamlessly with the MOM infrastructure.

The documentation and real examples based on Apache Camel can be found on the Apache Camel web site.

[29] Apache Camel - `http://camel.apache.org`
[30] Apache ServiceMix - `http://servicemix.apache.org`