EUROPEAN COMMISSION

DIGIT
Connecting Europe Facility

Domibus 3.3 RC1

Software Architecture Document

Date: 29/06/2017

Document Approver(s):

Approver Name	Role
Adrien FERIAL	IT Project officer

Document Reviewers:

Reviewer Name	Role
Cosmin BACIU	Developer

Summary of Changes:

Version	Date	Created by	Short Description of Changes
0.1	21.07.2016	Christian KOCH	Initial draft
0.2	27.09.2016	Christian KOCH, Stefan MÜLLER	Changes according to comments
0.3	30.09.2016	Stefan MÜLLER	Chapters about logging and caching added
0.4	05.10.2016	Federico Martini	Added information about TB_MESSAGE_LOG and TB_ERROR_LOG
0.5	06.10.2016	Christian KOCH, Stefan MÜLLER	Information about certificate handling and retry mechanism added. Graphics updated.
1.0	10.10.2016	Adrien FERIAL	Fist published version
1.1	13.01.2017	Cosmin BACIU	Describe the new Domibus logging framework
1.2	29.06.2017	Cosmin BACIU	Describe the domibus-ext-services-api

Table of Contents

1. INTRODUCTION

1.1. Purpose

This document provides a comprehensive architectural overview of the system, using a number of different architectural views to depict individual aspects of the system. It is intended to capture and convey the significant architectural decisions that have been made on the system.

1.2. Scope

The architecture described in this document concerns the middleware Domibus created by the e-CODEX project and adopted by the CEF as sample platform for e-delivery. It is compliant with the e-SENS profile [REF2] of the OASIS ebMS3/AS4 standard. This document is not intended to explain the ebMS3/AS4 standards, the four-corner model or any other concepts described in the provided references.

1.3. References

#	Document	Contents outline
[REF1]	e-SENS AS4 Profile	The e-SENS AS4 Profile is a profile of the ebMS3 and AS4 OASIS Standards. It has provisions for use in four-corner topologies, but it can also be used in point-to-point exchanges.
[REF2]	OASIS AS4 Profile	AS4 Profile of ebMS 3.0 Version 1.0. OASIS Standard, 23 January 2013.
[REF3]	ebMS3 Core	OASIS ebXML Messaging Services Version 3.0: Part 1, Core Features. OASIS Standard. 1 October 2007.
[REF4]	Domibus plugin cookbook	Technical manual on Domibus plugin development.
[REF5]	Apache CXF	Apache CXF is an open source services framework. These services can speak a variety of protocols such as SOAP, XML/HTTP, RESTful HTTP, or CORBA and work over a variety of transports such as HTTP, JMS or JBI.

[REF6]	Apache WSS4J	The Apache WSS4J™ project provides a Java implementation of the primary security standards for Web Services, namely the OASIS Web Services Security (WS-Security) specifications from the OASIS Web Services Security TC
[REF7]	WS-Policy Specification	The Web Services Framework provides a general-purpose model and corresponding syntax to describe the policies of entities in a Web services-based system.
[REF8]	Spring Security	Spring Security is a framework that focuses on providing both authentication and authorization to Java applications. It can easily be extended to meet custom requirements.
[REF9]	Bcrypt	*Provos, Niels; Mazières, David; Talan Jason Sutton 2012 (1999). "A Future-Adaptable Password Scheme". Proceedings of 1999 USENIX Annual Technical Conference: 81–92.*
[REF10]	JMS	The Java Message Service (JMS) API is a Java Message Oriented Middleware API for sending messages between two or more clients.
[REF11]	e-CODEX	The e-CODEX project improves the cross-border access of citizens and businesses to legal means in Europe and furthermore creates the interoperability between legal authorities within the EU.
[REF12]	Java Servlet 3.0	A Java servlet is a Java program that extends the capabilities of a server. Although servlets can respond to any types of requests, they most commonly implement applications hosted on Web servers. Such Web servlets are the Java counterpart to other dynamic Web content technologies such as PHP and ASP.NET.
[REF13]	Xtext	Xtext is a framework for development of programming languages and domain-specific languages.
[REF14]	SOAP	Simple Object Access Protocol

[REF15] HTTP Chunking	A mechanism by which data is broken up into a number of chunks when sent over an HTTP connection.

1.4. Document Content Overview

After summarizing the architectural representation, goals and constraints, this document describes the system using several architectural views (Use Case, logical, process, deployment, implementation and data) and then concludes with size, performance and quality considerations.

2. ARCHITECTURAL REPRESENTATION

The next two sections of the document describe the architectural goals and constraints.

Architecturally relevant Use Cases are described by a Use Case diagram and a short explanation of their impact on the architecture. The following views will also be provided:

- A logical view provides a high-level view of the platform presenting the structure of the system through its components and their interactions.
- An implementation view describes the software layers and the main software components. A component diagram is used in this view.
- A deployment view provides a description of the hardware components and how they are linked together. This view gives a technical description of protocols and hardware nodes used.
- A data view provides information about the data persistency. A class diagram will be used to model the main system data.

UML diagrams are systematically used to represent the different views of the system.

3. ARCHITECTURAL GOALS AND CONSTRAINTS

The following non-functional requirements that affect the architectural solution have been identified:

Non-functional requirement	Description
Adaptability	The application shall be easy to be integrated into existing business workflows using different communication protocols and data formats
Portability	The application shall be able to be deployed on a wide variety of software/hardware systems
Interoperability	The system shall be interoperable with both commercial and free alternative implementations of the e-SENS profile.

4. SECURITY

4.1. Introduction

The Domibus middleware provides built-in security in accordance to the implemented [REF2] specification and industry best practices. It can also be easily integrated into existing security domains.

4.2. Corner 1 - Corner 2 Communication

As no assumptions can be made about the security architecture of corner 1/4 (back office), the integration into the existing architecture has to be provided by the Domibus plugins. While the default plugins do not include any security constraints they can be easily extended to accommodate most of the security requirements.

4.3. Corner 2 – Corner 3 Communication

The communication between corner 2 and corner 3 is able to fulfil all the security requirements specified in the e-SENS AS4 profile. The configuration is handled via WS-Policy files and PMode configuration. All webservice security is enforced by the Apache CXF framework [REF5].

4.3.1. Certificate Configuration

The location and credentials of private and public certificates used by CXF are configured in the "domibus-security.xml" spring configuration file.

4.3.2. Client Certificate

The client certificate for use with client authentication (two-way SSL) is configured in the "clientauthentication.xml" spring configuration file. Incoming TLS secured connections terminate at the proxy server (e.g. Apache httpd) and must be configured according to the employed proxy servers documentation.

4.4. Corner 3 – Corner 4 Communication

The security between corner 3 and corner 4 is handled via the same mechanisms used in the communication corner 1 – corner 2.

4.5. Administrative Sites

Access to the domibus administration page is secured with username/password. The credentials are managed by a Spring authentication manager and multiple authentication providers can be plugged into it By default, the credentials are stored in the Domibus-security.xml file and they are managed by an authentication provider that uses a Bcrypt strong hashing function for encoding them. Integration into an existing authentication scheme (i.e. LDAP) can be performed via Spring configuration.

SECURITY DISCLAIMER

On top of the security that Domibus provides, the user shall take additional security measures according to best practices and regulations. This includes, but is not limited to, using firewalls, IP whitelists and file system/database encryption. DIGIT shall not be held responsible for any security breach that might occur due to User not respecting this recommendation.

5. USE-CASE VIEW

This section provides a representation of the use cases relevant for the architecture.

5.1. Selection Rationale

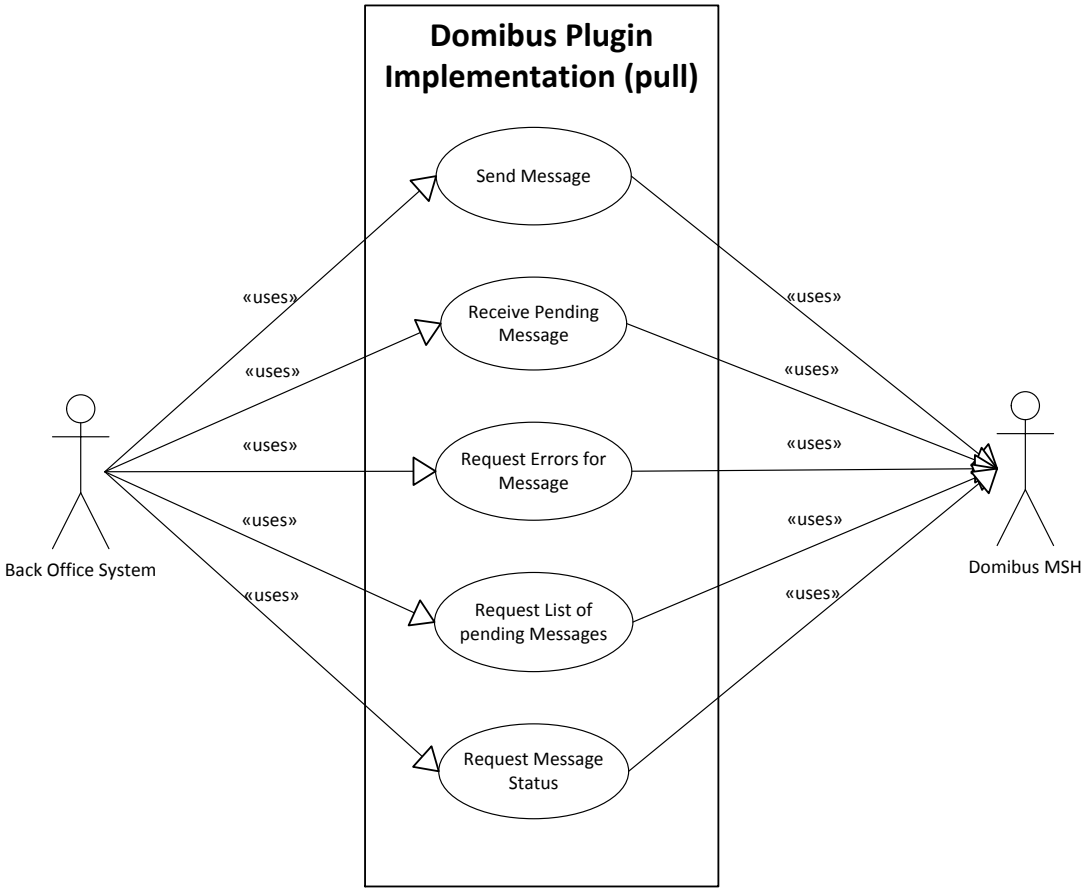
The uses cases relevant for the architecture have been selected based on the following criteria:

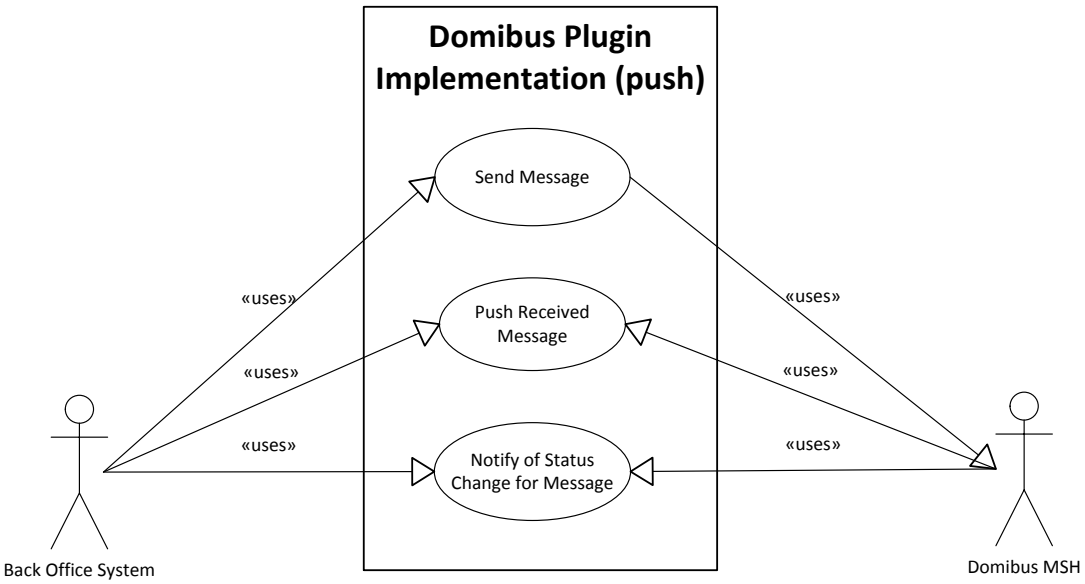
– Use cases affecting the exchange between the back office system and the Domibus MSH.

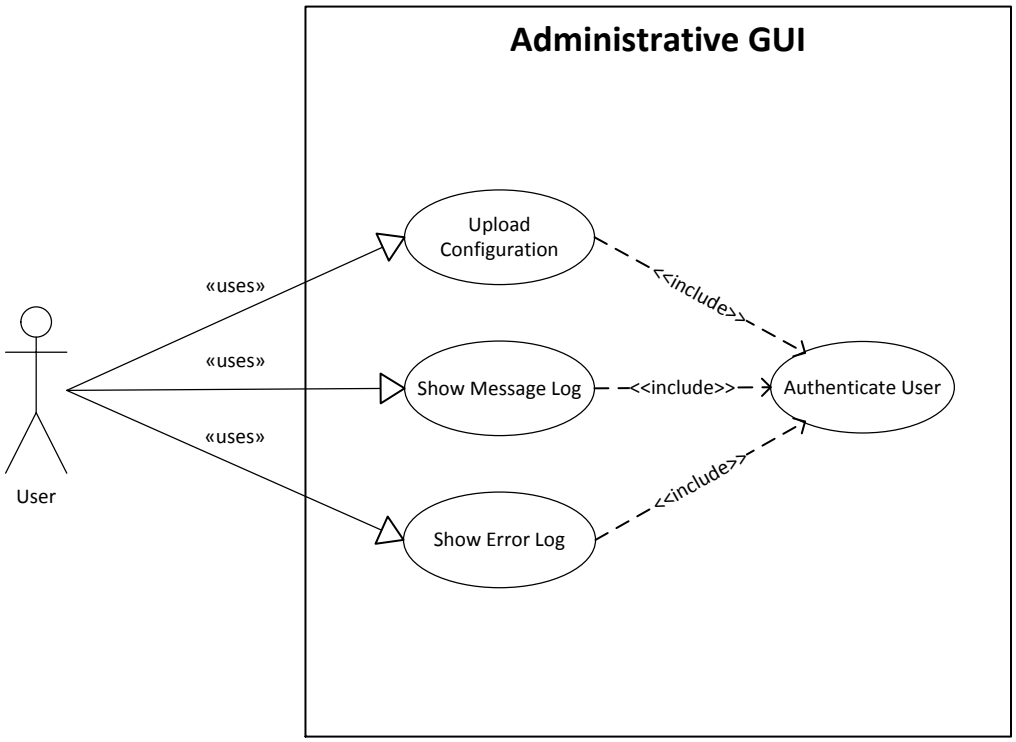
– Use cases representing critical parts of the architecture, thereby addressing the technical risks of the project at an earlier stage.

The following use cases have been selected:

- Back office integrations using pull communication (i.e. Webservice)
- Back office integrations using push communication (i.e. JMS)
- Usage of the administrative GUI




Domibus Plugin Implementation (push)
Send Message
Push Received Message
Notify of Status Change for Message
«uses»
«uses»
«uses»
«uses»
«uses»
«uses»
Back Office System
Domibus MSH


Administrative GUI
Upload Configuration
Show Message Log
Show Error Log
Authenticate User
<<include>>
<<include>>
<<include>>
«uses»
«uses»
«uses»
User

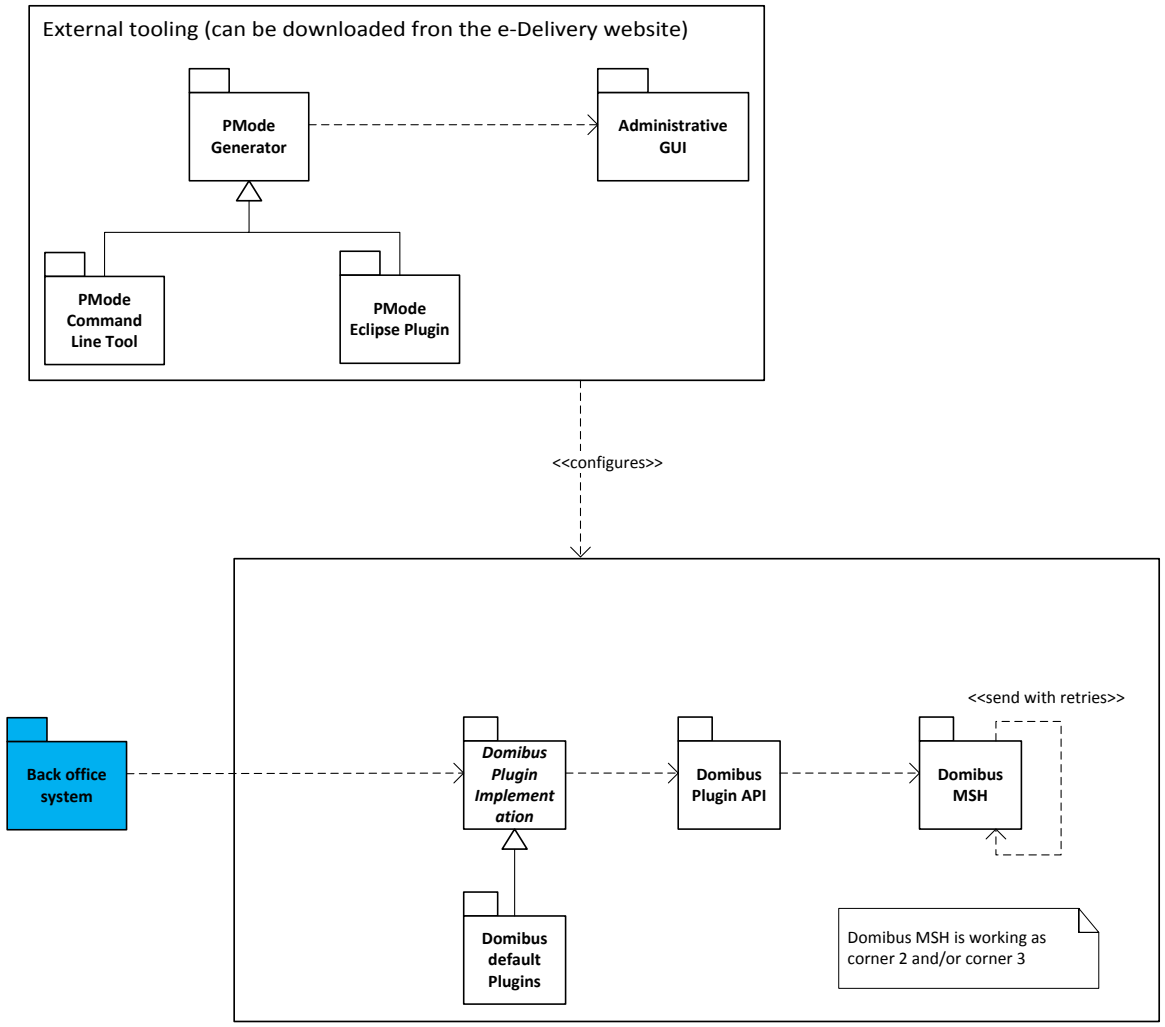
6. LOGICAL VIEW

6.1. Overview

This chapter describes the main application modules, how they interact and how they implement the specification and profile.

6.2. Architecturally Significant Design Packages

The following diagram provides a high-level view of the main packages composing the system. The Database persistence and file system persistence are logical packages representing the physical data storages used by the platform. The others represent different application layers and give an overview of the organisation of the platform's code.



6.2.1. *Back office system (Corner 1/4)*

The whole purpose of Domibus is to connect different back office systems via structured, secure message exchange. While, regarding a single message exchange, corner 1 and 4 are usually different applications running in different environments, within a single deployment the role of corner 1 and corner 4 (for different message exchanges) is usually occupied by the same application. Therefore, from a logical point of view, corner 1 and 4 are the same package.

6.2.2. Domibus plugin implementation

This module is responsible for the communication between the back office system and Domibus and for the mapping from the back office internal data format to Domibus internal data format. The communication and the mapping of the data can be done in both directions. Integration into existing security architecture can also be implemented here.

As there can be made few assumptions about the back office system, this module is commonly implemented by the Domibus user. Details on this process can be found inside the Domibus plugin cookbook.

6.2.3. Domibus default plugins

Domibus provides two default plugins, which serve for testing purposes and as examples for custom implementations. They were initially developed to accommodate the needs of the e-CODEX project and thus will not be suitable for every use case.

6.2.4. Domibus plugin API

This package contains all necessary interfaces and classes required to implement a Domibus plugin

6.2.5. Domibus MSH (Corner 2/3)

The Domibus MSH (Message Service Handler) is the main module, representing corner 2 and/or 3 in a 4-corner message exchange. All the implementation relevant to the e-SENS profile is done inside this package. It is deployable on any Container supporting the Java Servlet Specification v 3.0.

To support the required AS4 retry mechanisms a spring configured cronjob regularly checks for messages that need to be resent. The cronjob is configured using spring (domibus-configuration.xml) on property: “domibus.msh.retry.cron”. This does not configure the retry interval for messages (which is done via PModes).

6.2.6. Administrative GUI

This package contains of a Spring MVC web application providing basic monitoring and configuration options.

6.2.7. PMode generator

The PMode generator is an external tool which is able to create a set of XML PMode configuration files (one for each Domibus MSH in the network) using a proprietary domain specific language (pconf). This generator is implemented with Xtext.

6.2.8. PMode command line tool

The PMode command line tool is a maven project which is able to generate XML PMode configuration files from a pconf file via command line

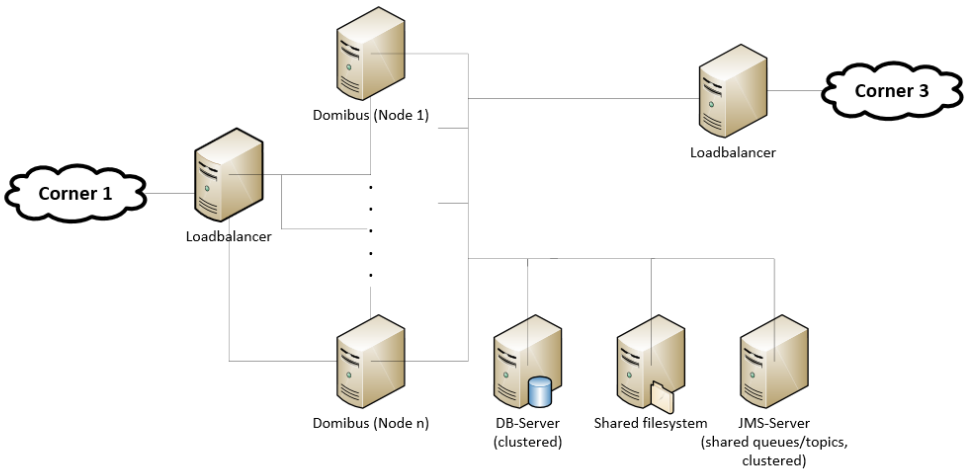
6.2.9. PMode Eclipse plugin

The PMode Eclipse plugin is an extension to the open source Eclipse development environment. It provides editing capabilities for pconf files including code completion and syntax highlighting.

7. DEPLOYMENT VIEW

The following is a description of the hardware nodes running the execution environment for the system.

The following diagram provides a view of hardware components involved in this project. Note that a clustered environment is shown. If a single server deployment is sufficient (i.e. for testing purposes), a load balancer and multiple hardware nodes are not required.



It is important to note that not all physical nodes are represented on this diagram. Indeed load balancers, database servers and JMS servers could be duplicated for scalability, performance and availability reasons. Furthermore, security mechanisms like firewalls are not shown.

These are the identified hardware nodes.

- Load balancers are responsible for distributing requests among multiple Domibus nodes. A random round robbing/no sticky session setup is recommended.
- Java servlet containers with deployed Domibus instances are responsible for message processing
- A database server (MySQL 5.6+ or Oracle 11g+) is responsible for storing messages and PMode configuration data
- The shared file system contains shared Domibus configuration data, file based PMode data (Keystores) and, depending on configuration, binary data of message attachments.

Domibus has been successfully tested on Tomcat 8, WebLogic 12.1.x and WildFly 9.0.2 servers.

8. IMPLEMENTATION VIEW

8.1. Overview

The following diagram describes the software layers of the system and their components.

The AS4 MSH Service is the web service which accepts the AS4 requests and it is the one that is called by the external systems. External MSH services are accessed through the **AS4 Message Dispatch Service**, which is a web service client capable of sending AS4 requests. The Back office systems can access the platform through their respective plugin implementations. The **Web Layer** is accessed typically by a web browser. The MSH SOAP handling is implemented using the Apache CXF framework.

The **Integration Layer** uses the Spring framework and is responsible for the integration of custom plugins and all communication processes and data format translations between back office systems and Domibus.

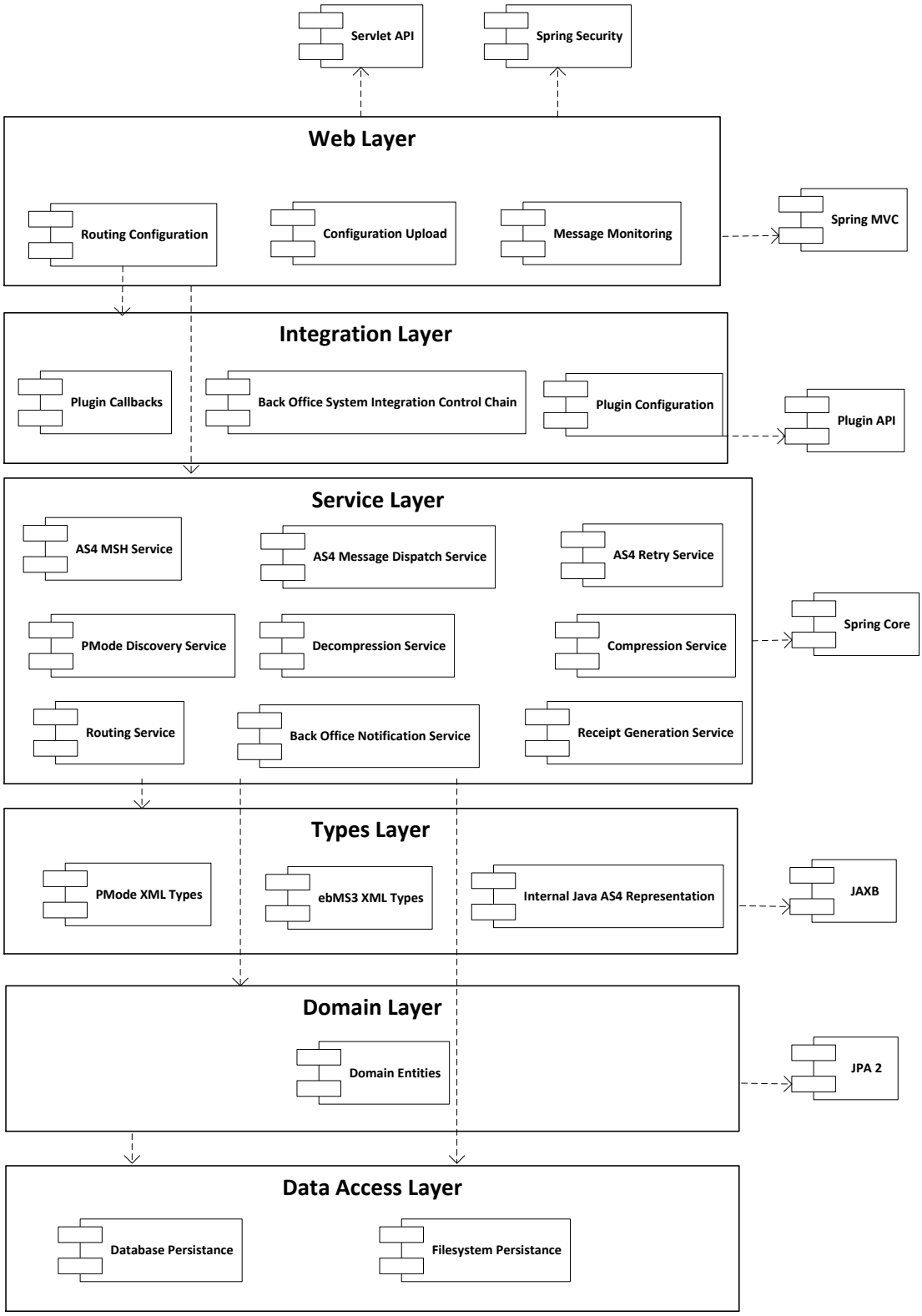
The **Services Layer** offers access to the domain objects of the platform as well as to the platform data layer. These services are Plain Old Java Objects relying on the Spring framework for dependency injection and for transaction management.

The **Types Layer** contains all the java objects generated from the XSDs used by the platform. These are JAXB generated objects.

The **Domain Layer** holds all the platform entities. The persistence of these entities is implemented using the Java Persistence API version 2.0.

Finally, the **Data Persistence** relies on the database and the file system to persist the data. The file system is used to store configuration data and the database to persist the incoming and outgoing messages.

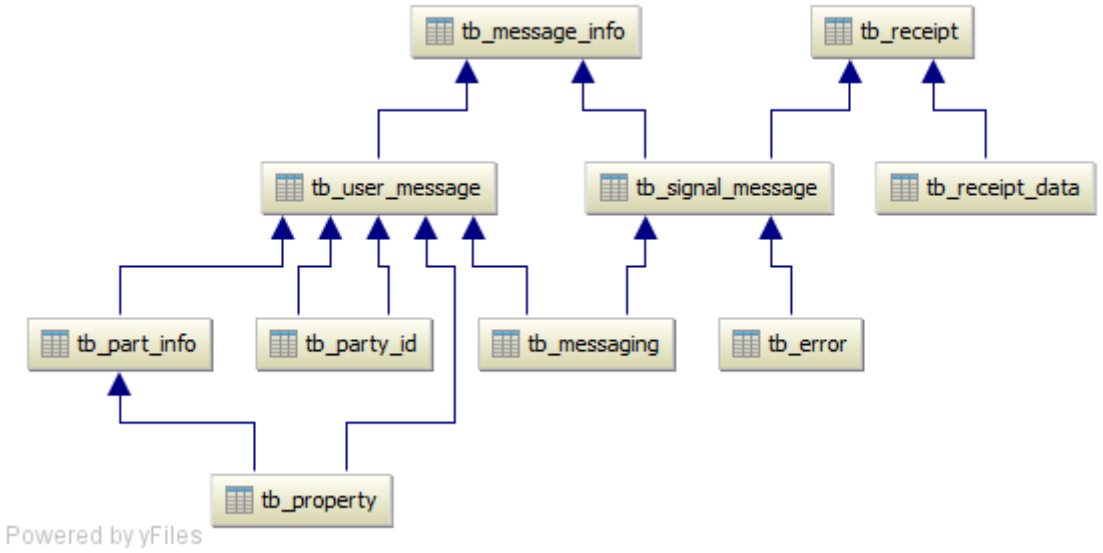
All these layers run on a Java Servlet Container. The platform has been tested on Tomcat 8.x, WildFly 9.0.2 and WebLogic 12.1.x.


Servlet API
Spring Security
Web Layer
Routing Configuration
Configuration Upload
Message Monitoring
Spring MVC
Integration Layer
Plugin Callbacks
Back Office System Integration Control Chain
Plugin Configuration
Plugin API
Service Layer
AS4 MSH Service
AS4 Message Dispatch Service
AS4 Retry Service
PMode Discovery Service
Decompression Service
Compression Service
Spring Core
Routing Service
Back Office Notification Service
Receipt Generation Service
Types Layer
PMode XML Types
ebMS3 XML Types
Internal Java AS4 Representation
JAXB
Domain Layer
Domain Entities
JPA 2
Data Access Layer
Database Persistance
Filesystem Persistance

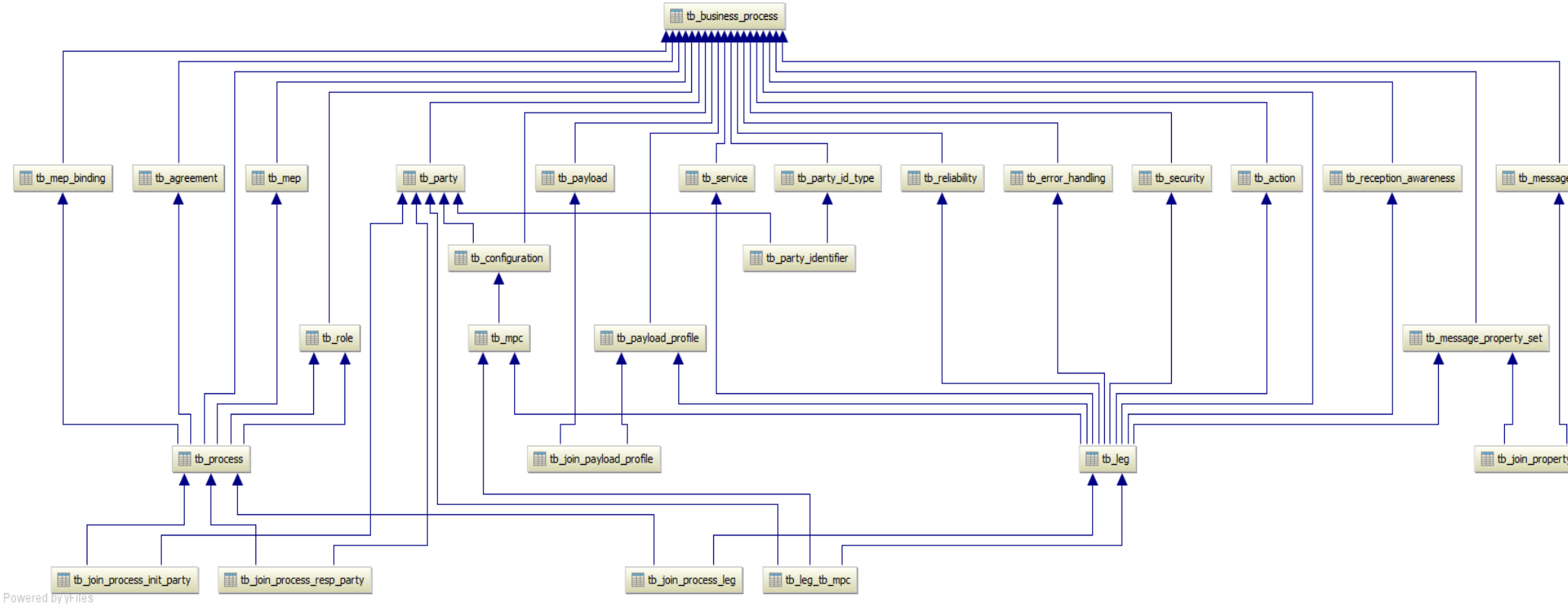
9. DATA VIEW

9.1. Data Model

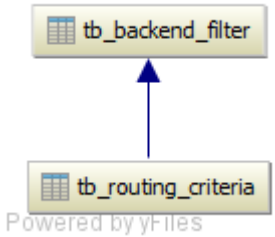
The following diagrams show a high-level abstraction of the data entities, which must be implemented by the system:



The above tables represent a 1:1 mapping of the ebMS3 XSD to database tables.

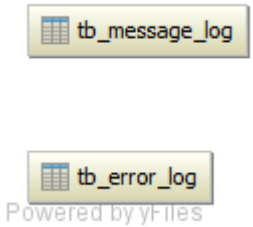

tb_business_process
tb_mep_binding
tb_agreement
tb_mep
tb_party
tb_payload
tb_service
tb_party_id_type
tb_reliability
tb_error_handling
tb_security
tb_action
tb_reception_awareness
tb_configuration
tb_party_identifier
tb_role
tb_mpc
tb_payload_profile
tb_message_property_set
tb_process
tb_join_payload_profile
tb_leg
tb_join_process_init_party
tb_join_process_resp_party
tb_join_process_leg
tb_leg_tb_mpc
Powered by yFiles

The above tables represent a 1:1 mapping of the PMode configuration XSD to database tables.



Routing criteria contains the data that are needed to perform the routing of the messages to a specific plugin implementation.

Backend filters are collections of routing criteria associated with a specific backend representation.



The TB_MESSAGE_LOG table contains information about the User Messages and the Signal Messages (both sent and received ones). The stored values are the following:

MESSAGE_ID, MESSAGE_STATUS, MESSAGE_TYPE, MPC, MSH_ROLE, NEXT_ATTEMPT, NOTIFICATION_STATUS, RECEIVED, SEND_ATTEMPTS, SEND_ATTEMPTS_MAX, BACKEND, ENDPOINT, DELETED

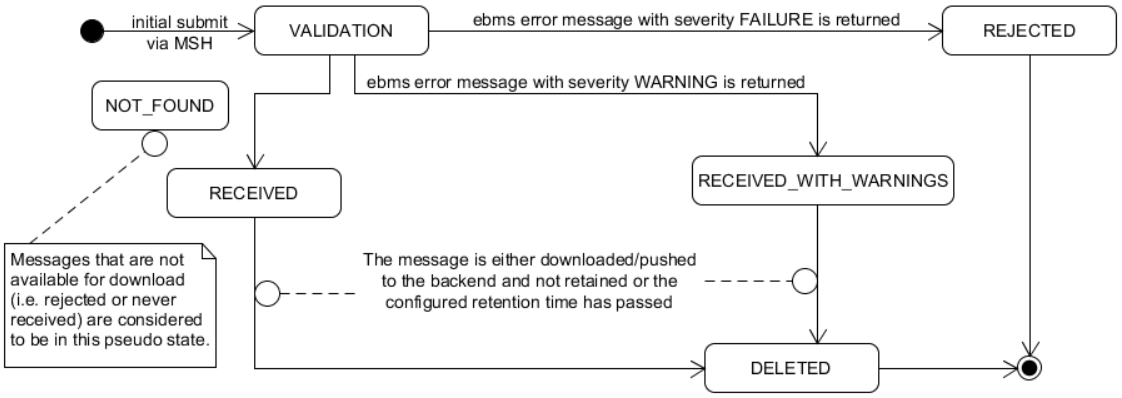
The TB_ERROR_LOG table contains information of the errors occurred during message transmission. The stored values are the following:

ERROR_CODE, ERROR_DETAIL, ERROR_SIGNAL_MESSAGE_ID, MESSAGE_IN_ERROR_ID, MSH_ROLE, NOTIFIED, TIME_STAMP

9.2. State Machines

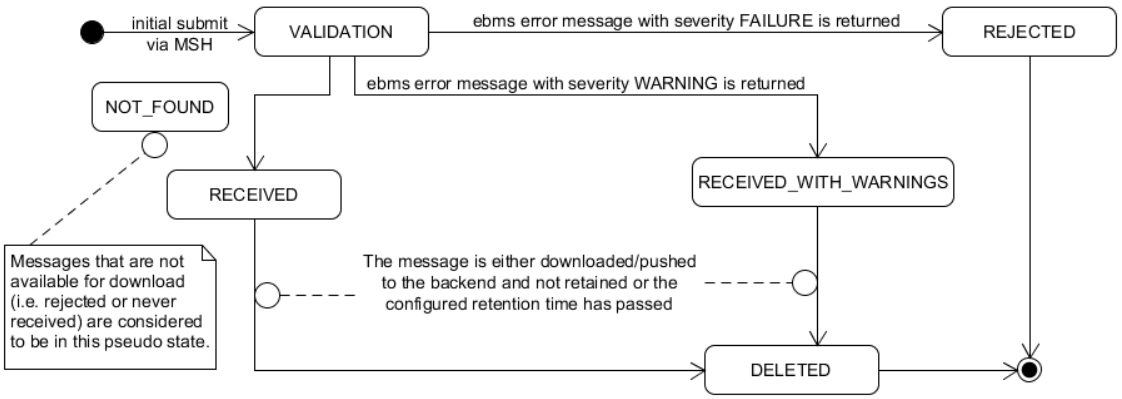
9.2.1. Outgoing Message State Machine

The outgoing messages have the following state machine:



9.2.2. Incoming Message State Machine

The incoming messages have the following state machine:



10. SIZE AND PERFORMANCE

10.1. Size

Size restrictions applied on the data that is exchanged by the back office systems, but not on the application or its components themselves, have an impact on the architecture and on the configuration of the system.

To support the exchange of large binary files, the plugin API supports payload submission by reference, meaning that Domibus is able to download a payload from a given URI. Additionally payloads can be stored on the file system instead of the database to avoid the processing of huge blobs.

As the e-SENS AS4 profile provides no provisions for ebMS large file handling (split/join) the transfer of data is limited by bandwidth and memory constraints.

Extra restrictions can be implemented via the business process PModes. These restrictions concern the maximum size of a payload and the maximum number of payloads in a message.

10.2. Performance

An important architectural decision that benefits the performance of Domibus includes the decoupling of the solution into corner 1/4 representing the back office systems and corner 2/3 representing the Domibus MSH.

The back office systems (corner1/4) interact with the Domibus MSH (corner 2/3) via the interfaces (web services, JMS, REST, etc) exposed by the plugins deployed on the Domibus MSH side.

Domibus MSH is using internally JMS queues to perform the processing of the messages coming from the back office systems via the plugins or from other access points.

All this architectural decisions lead to an improved throughput and load distribution of the messages.

11. QUALITY

The architecture of Domibus contributes to quality aspects of extensibility, reliability and portability in the following ways.

11.1. Extensibility

Domibus is designed in a layered fashion and consists of multiple interconnected modules. This modular design facilitates the upgrades by replacing existing modules and extensions by adding additional modules.

11.2. Reliability

The reliability of Domibus is enhanced through the decoupling of each architectural layer by JMS queues. A store and forward mechanism and automatic retry policy ensures that parts of the system can continue functioning without losing data when an issue occurs in a specific component.

11.3. Portability

Currently the application can be deployed on Tomcat 8, WebLogic 12.1.x and WildFly 9.0.2 and can connect to Oracle and MySQL databases.

With minor changes, it might be deployed on any Java Servlet 3.0-compliant server and it might connect to any RDBMS (Relational Database Management System).

Besides being extensible, Domibus is carefully designed in such a way that it is independent of the specific external system that communicates with. The use of a generic plugin API leaves the different layers unaffected when an additional external systems need to be supported by Domibus.

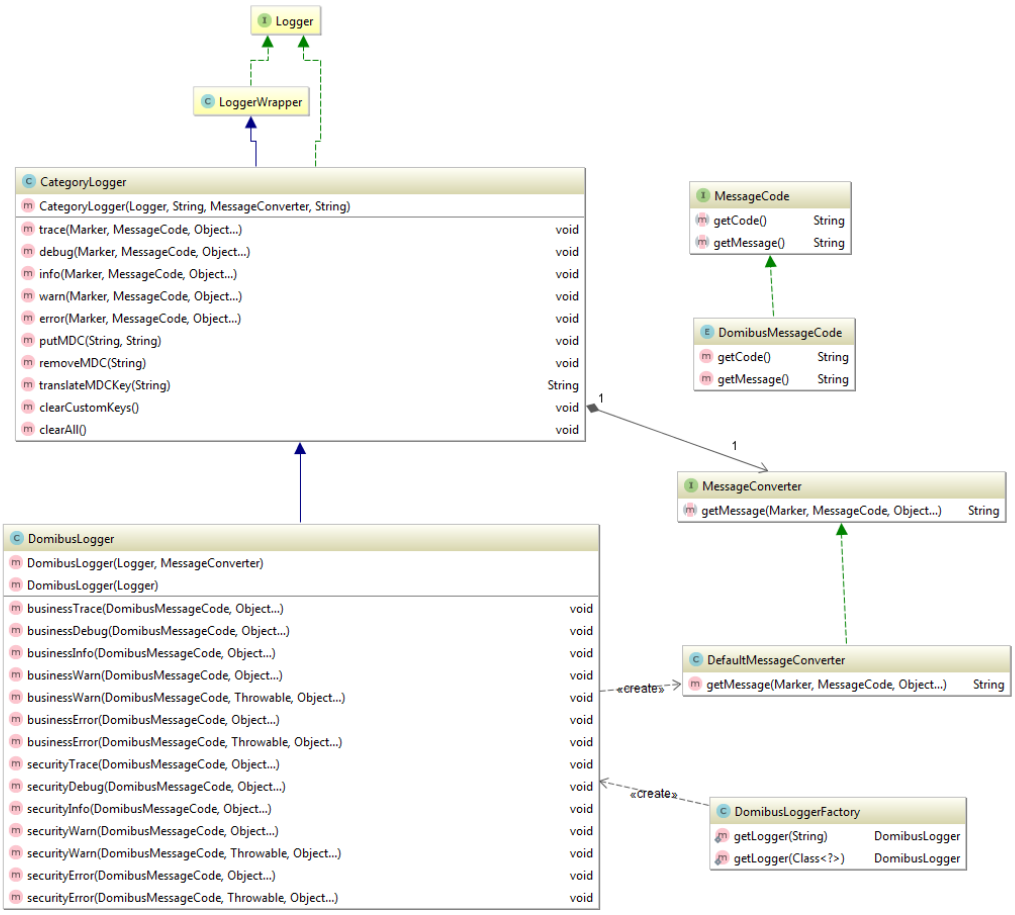
The usage of JPA to access the database makes it easy for implementers to change the relational database used to store the platform data.

12. Logging

12.1. Implementation

The logging framework used by Domibus is SLF4J API together with Logback as the SLF4j implementation.

The *domibus-logging* module provides the custom SLF4J logger *DomibusLogger*. This logger must be used for all the logs within the Domibus application.



There are three types of logs: security logs, business logs and miscellaneous logs. Each log category has its own marker defined in the DomibusLogger class. By default, each category will be logged in a separate file:

- domibus-security.log : This log file contains all the security related information. For example, you can find information about the clients who connect to the application.
- domibus-business.log: This log file contains all the business related information. For example, when a message is sent or received, etc.
- domibus.log : This log file contains both the security and business logs plus miscellaneous logs like debug information, logs from one of the framework used by the application, etc.

The security and business logs require a code that is defined in the *DomibusMessageCode* class.

The logs pattern is defined in the *logback.xml* file. The default pattern is:

```
%d{ISO8601} [%X{d_user}] [%X{d_messageId}] %5p %c{1}:%L - %m%n
```

- *d_user*: The authenticated user.
- *d_messageId*: The message id currently being sent/received.

The values for the *d_user* and *d_messageId* properties can be set by calling the method *DomibusLogger.putMDC(String key, String value).* The prefix ***d_*** is added automatically by the DomibusLogger in order to easily identity the Domibus specific MDC properties.

Eg:

```
LOGGER.putMDC(DomibusLogger.MDC_USER, authenticationResult.getName());
```

The MDC values need to be always cleaned after the thread execution. Otherwise, the thread might be returned back to the thread pool with previously set MDC values and on the next thread execution, the old MDC values will be used.

In order to easily clear the MDC values after a method execution a custom annotation, *MDCKey*, has been created in order to mark a method that is setting values in the MDC. An AOP aspect is detecting the methods annotated with the *MDCKey* annotation and after the execution of the method it is clearing the MDC values.

Eg:

```
@MDCKey(DomibusLogger.MDC_MESSAGE_ID)
public String submit(final Submission messageData, final String backendName)
```

12.2. Domibus log codes

Category	Event code	Description
SECURITY	SEC-001	Unsecure login is allowed, no authentication will be performed
SECURITY	SEC-002	Basic authentication is used
SECURITY	SEC-003	X509Certificate authentication is used
SECURITY	SEC-004	Blue coat authentication is used
SECURITY	SEC-005	The host [{}] attempted to access [{}]
SECURITY	SEC-006	The host [{}] has been granted access to [{}] with roles [{}]
SECURITY	SEC-007	The host [{}] has been refused access to [{}]

SECURITY	SEC-008	Certificate is not valid at the current date [{}]. Certificate valid from [{}] to [{}]
SECURITY	SEC-009	Certificate is not yet valid at the current date [{}]. Certificate valid from [{}] to [{}]
BUSINESS	BUS-001	Message successfully received
BUSINESS	BUS-002	Failed to receive message
BUSINESS	BUS-003	Failed to validate message
BUSINESS	BUS-004	Failed to notify backend for incoming message
BUSINESS	BUS-005	Invalid charset [{}] used
BUSINESS	BUS-006	Invalid NonRepudiationInformation: no security header found
BUSINESS	BUS-007	Invalid NonRepudiationInformation: multiple security headers found
BUSINESS	BUS-008	Invalid NonRepudiationInformation: eb:Messaging not signed
BUSINESS	BUS-009	Invalid NonRepudiationInformation: non repudiation information and request message do not match
BUSINESS	BUS-010	There is no content inside the receipt element received by the responding gateway
BUSINESS	BUS-011	Reliability check failed, check your configuration
BUSINESS	BUS-012	Reliability check was successful
BUSINESS	BUS-013	Compression failure: no mime type found for payload with cid [{}]
BUSINESS	BUS-014	Error compressing payload with cid [{}]
BUSINESS	BUS-015	Payload with cid [{}] has been compressed

BUSINESS	BUS-016	Decompression failure: no mime type found for payload with cid [{}]
BUSINESS	BUS-017	Payload with cid [{}] will be decompressed
BUSINESS	BUS-018	Decompression is not performed: leg compressPayloads parameter is false
BUSINESS	BUS-019	Decompression is not performed: partInfo with cid [{}] is in body
BUSINESS	BUS-020	Message action [{}] found for value [{}]
BUSINESS	BUS-021	Message action not found for value [{}]
BUSINESS	BUS-022	Message agreement [{}] found for value [{}]
BUSINESS	BUS-023	Message agreement not found for value [{}]
BUSINESS	BUS-024	Party id [{}] found for value [{}]
BUSINESS	BUS-025	Party id not found for value [{}]
BUSINESS	BUS-026	Party [{}] is not a valid URI [CORE] 5.2.2.3
BUSINESS	BUS-027	Message service [{}] found for value [{}]
BUSINESS	BUS-028	Message service not found for value [{}]
BUSINESS	BUS-029	Message service [{}] is not a valid URI [CORE] 5.2.2.8
BUSINESS	BUS-030	Leg name found [{}] for agreement [{}], senderParty [{}], receiverParty [{}], service [{}] and action [{}]
BUSINESS	BUS-031	Leg name not found found for agreement [{}], senderParty [{}], receiverParty [{}], service [{}] and action [{}]
BUSINESS	BUS-032	Preparing to send message

BUSINESS	BUS-033	Message sent successfully
BUSINESS	BUS-034	Message send failure
BUSINESS	BUS-035	No Attachment found for cid [{}]
BUSINESS	BUS-036	More than one Partinfo referencing the soap body found
BUSINESS	BUS-037	Payload profile validation skipped: payload profile is not defined for leg [{}]
BUSINESS	BUS-038	Payload profiling for this exchange does not include a payload with CID [{}]
BUSINESS	BUS-039	Payload profiling for this exchange requires all message parts to declare a MimeType property [{}]
BUSINESS	BUS-040	Payload profiling error, missing payload [{}]
BUSINESS	BUS-041	Payload profile [{}] validated
BUSINESS	BUS-042	Property profile validation skipped: property profile is not defined for leg [{}]
BUSINESS	BUS-043	Property profiling for this exchange does not include a property named [{}]
BUSINESS	BUS-044	Property profile [{}] validated
BUSINESS	BUS-045	Message persisted
BUSINESS	BUS-046	Message receipt generated with nonRepudiation value [{}]
BUSINESS	BUS-047	Message receipt generation failure
BUSINESS	BUS-048	Message status updated to [{}]
BUSINESS	BUS-049	All payloads data for user message [{}] have been cleared

BUSINESS	BUS-050	Security policy [{}] was not found for outgoing message
BUSINESS	BUS-051	Security policy [{}] is used for outgoing message
BUSINESS	BUS-052	Security algorithm [{}] is used for outgoing message
BUSINESS	BUS-053	Security algorithm [{}] is used for incoming message
BUSINESS	BUS-054	Security encryption username [{}] is used for outgoing message
BUSINESS	BUS-055	Security policy [{}] for incoming message was not found
BUSINESS	BUS-056	Security policy [{}] for incoming message is used

13. CACHING

In order to enhance the performance domibus uses caching in specific areas of the application:

- caching of security policies
- caching of backend filter configuration
- caching of pmodes, when using the “CachingPModeProvider”

This is the default configuration of ehcache defines the following properties:

```
maxBytesLocalHeap = 5m
timeToLiveSeconds = 3600
overflowToDisk= false
```

14. EXTERNAL API

Before the introduction of the external API module, the API that was exposed to the plugin implementers was strictly coupled with the internal core. If the signature of one of methods from the API changed, it affected both the plugin implementers and the internal core. There was no easy way to guarantee backward compatibility of the API to the plugin implementers.

The goal of the external API module is to tackle the two issues explained above: guarantee the backward compatibility to the plugin implementers and decouple the internal core from the external API.

The external API will not change so frequently as Domibus. Therefore, it makes sense that the external API has different lifecycle than the one of the Domibus lifecycle and it can evolve independently. Technically this means that a new release of Domibus does not necessarily mean a new release of the external API.

The external API functionality can be accessed in two ways:

- Java API

It can be used by the plugin implementers in the custom plugins. The following Maven dependency is necessary in order to have access to the external API:

```
<dependency>
    <groupId>eu.domibus</groupId>
    <artifactId>domibus-ext-services-api</artifactId>
    <version>${domibus-ext-services-api.version}</version>
</dependency>
```

- REST interface

The REST interface can be used directly by the C1/C4 backends if the network configuration allows it.

14.1. Message acknowledgement service

This service is used to acknowledge when a message is:

- delivered from C3 to the backend
- processed by the backend

Here are the typical use cases for using the *MessageAcknowledgementService*:

- a message is received by C3 from C2: the plugin that handles the message registers an acknowledgment before delivering the message to the backend

- a message is processed by the backend and it informs C3 via the plugin; the plugin registers an acknowledgment that the message has been processed by the backend

- a message is processed by the backend and informs C3 directly via the REST service exposed by the core; a REST service is exposed containing the same signature as {@link MessageAcknowledgeService}

There are two ways of performing message acknowledgments between C3 and the backend:

- synchronous

C3(via the plugin) notifies the backend synchronously and the backend process the messages also synchronously. In this case, there is no need for the backend to send a separate message acknowledgement so the plugin at the C3 side registers the processing of the message by the backend.

```
Eg:
BackendResponse backendResponse = plugin.callBackendWS(message)
messageAcknowledgeService.acknowledgeMessageDelivered(message.getId(),
new Timestamp(System.currentTimeMillis()))
messageAcknowledgeService.acknowledgeMessageProcessed(message.getId(),
new Timestamp(System.currentTimeMillis()))
```

- asynchronous

C3 notifies the backend synchronously and the backend process the messages asynchronously. In this case, the backend will send a separate message acknowledgement when it manages to process the message successfully.

```
Eg:
plugin.sendMessageToTheBackend(message)
messageAcknowledgeService.acknowledgeMessageDelivered(message.getId(),
new Timestamp(System.currentTimeMillis()))
```

14.2. Monitoring service

This service is used to monitor failed messages and to restore them if necessary.

Assuming that "failed message" means failed to be sent by the sender access point and getting the status set to SEND_FAILURE, the service gives the possibility to:

- list all the failed messages

- restore a failed message

- restore all messages failed during a specific period

- know how long time a message has been failed

- get the history of all delivery attempts

- delete the message payload of a failed message

14.3. Architecture

To offer a better picture of the architecture that has been implemented you can find below as an example the class diagram of one of the services exposed, the message acknowledgment service.