



EUROPEAN COMMISSION

DIGIT
Digital Europe Programme

Software Architecture Document

© European Union, 2023

Reuse of this document is authorised provided the source is acknowledged. The Commission's reuse policy is implemented by Commission Decision 2011/833/EU of 12 December 2011 on the reuse of Commission documents.

Date: 18/09/2023

Document Approver(s):

Approver Name	Role
Bogdan DUMITRIU	eDelivery Project Officer

Document Reviewers:

Reviewer Name	Role
Cosmin BACIU	Lead Architect
Ioana Dragusanu	Technical leader
Caroline AEBY and Chaouki BERRAH	Technical Writer
Catalin-Emanuel ENACHE	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 FERIAI	First published version
1.1	13.01.2017	Cosmin BACIU	Documented new Domibus logging framework
1.2	29.06.2017	Cosmin BACIU	Documented the domibus-ext-services-api
1.3	03.10.2017	Cosmin BACIU	Documented the messageStatusChanged method and updated the diagrams for the outgoing/incoming message flows
1.4	09.10.2017	CEF Support	Update list of reviewers
1.5	24/11/2017	CEF Support	Updates for Domibus 3.3.1
1.6	03/01/2018	Tiago MIGUEL	Update Log Codes for Domibus 3.3.2
1.7	20/03/2018	CEF Support	Reuse notice added, e-Sens AS4 profile replaced by eDelivery AS4 profile
1.8	21/06/2018	CEF Support	Updated for Domibus 3.3.4 link to REST api)
1.9	30/07/2018	Cosmin BACIU & CEF Support	Documented multi-tenancy; moved the External API documentation in the Plugin Cookbook
1.10	04/09/2018	CEF Support	Domibus 4.0
1.11	17/09/2018	Caroline AEBY	Multi-tenancy => Multitenancy
1.12	18/09/2018	Chaouki BERRAH Catalin-Emanuel ENACHE	Update UIReplication added
1.13	26/09/2018	Caroline AEBY	CEF Support contact information added
1.14	07/12/2018	Caroline AEBY	Domibus 4.0.1 update
1.15	11/02/2019	Caroline AEBY	Domibus 4.0.2 update
1.16	16/04/2019	Caroline AEBY	Domibus 4.1-RC1 update
1.17	15/07/2019	Caroline AEBY	Version 4.1-RC1 => 4.1
1.18	09/09/2019	Caroline AEBY	Domibus 4.1 => 4.1.1

1.19	29/11/2019	Caroline AEBY	Domibus 4.1.1 => 4.1.2
1.2	04/02/2020	Caroline AEBY	Domibus 4.1.2 => 4.1.3
1.3	04/09/2020	Caroline AEBY	Domibus 4.1.5 + Domibus version agnostic
1.4	10/09/2020	Caroline AEBY Ion PERPEGEL Sebatian TINCU Arun VENUGOPAL	Domibus 4.2: Section 9.1.1 Data auditing added Section 14.5: properties with multimode Domibus log codes added
1.5	21/09/2020	Caroline AEBY	Domibus 4.2 RC version release
1.6	21/10/2020	Arun VENUGOPAL Caroline AEBY	Domibus 4.2. FR – updates in BUS log codes
1.7	25/02/2021	Caroline AEBY	Domibus 4.2.1 version
1.8	25/02/2021	Caroline AEBY	Domibus 4.2.2 version
1.8	29/04/2022	Caroline AEBY	Domibus 4.2.9 version + no more CEF references +links update
2.0	29/04/2022	Caroline AEBY	Domibus 5.0 version
3.0	18/10/2022	Caroline AEBY Gabriel MAIER	Logging separation in Multitenancy
3.1	15/11/2022	Caroline AEBY	Removed chapter on UI Replication (feature removed in Domibus 5.0)
3.2	22/11/2022	Caroline AEBY Gabriel MAIER	Information on loggers from third party libraries
3.3	25/11/2022	Caroline AEBY Cosmin BACIU	Documented distributed cache; added information about error.log
3.4	13/01/2023	Caroline AEBY Cosmin BACIU	Updated diagrams for the outgoing and incoming UserMessage exchange; updated the data view; other minor modifications
3.5	28/04/2023	Caroline AEBY Cosmin BACIU	Changes valid since for 5.0.5 version.
3.6	31/07/2023	Caroline AEBY Cosmin BACIU	Added diagram for message state pull flow
3.7	18/09/2023	Caroline AEBY Cosmin BACIU	Updated the Domibus Admin Console security description

Table of Contents

1. INTRODUCTION	6
1.1. Purpose.....	6
1.2. Scope	6
1.3. References.....	6
1.4. Document Content Overview.....	8
2. ARCHITECTURAL REPRESENTATION	9
3. ARCHITECTURAL GOALS AND CONSTRAINTS	10
4. SECURITY.....	11
4.1. Introduction.....	11

4.2. Corner 1 - Corner 2 Communication	11
4.3. Corner 2 – Corner 3 Communication	11
4.3.1. Certificate Configuration	11
4.3.2. Client Certificate.....	11
4.4. Corner 3 – Corner 4 Communication	11
4.5. Domibus Admin console.....	11
5. USE-CASE VIEW.....	13
6. LOGICAL VIEW	15
6.1. Overview	15
6.2. Architecturally Significant Components.....	15
6.2.1. Back office system (Corner 1/4)	16
6.2.2. Domibus plugin implementation.....	16
6.2.3. Domibus default plugins	16
6.2.4. Domibus plugin API	16
6.2.5. Domibus MSH (Corner 2/3).....	16
6.2.6. Administrative GUI	17
7. DEPLOYMENT VIEW	18
8. IMPLEMENTATION VIEW.....	19
8.1. Overview	19
9. DATA VIEW.....	21
9.1. Data Model.....	21
9.1.1. Data Auditing.....	24
9.2. State Machines.....	24
9.2.1. Outgoing Message State Machine	24
9.2.2. Incoming Message State Machine	24
9.2.3. Message State Machine for pull messages	25
10. SIZE AND PERFORMANCE	26
10.1. Size	26
10.2. Performance.....	26
11. QUALITY	27
11.1. Extensibility	27
11.2. Reliability.....	27
11.3. Portability	27
12. LOGGING	28
12.1. Implementation.....	28
12.2. Domibus log codes	29
12.3. Logging and multitenancy.....	35

13. CACHING	37
13.1. Local cache	37
13.2. Distributed cache	37
14. MULTITENANCY	38
14.1. General	38
14.2. Identifying the domain (tenant)	38
14.2.1. Selecting the database schema	39
14.3. User association to a domain	39
14.4. UI	39
14.4.1. Managing multiple domains from the Domibus Administration Console	39
14.4.2. Security	40
14.5. Plugins	40
14.5.1. Security	40
14.5.2. Plugin API	41
14.5.3. WS Plugin	41
14.5.4. JMS Plugin	41
14.5.5. FS Plugin	42
14.6. Domibus Properties	42
14.7. Message Payloads	44
14.8. Quartz	44
15. CONTACT INFORMATION	45

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 Domibus Access Point. It is compliant with the eDelivery 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]	eDelivery AS4 Profile	The eDelivery 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	<i>Provos, Niels; Mazières, David; Talan Jason Sutton 2012 (1999). "A Future-Adaptable Password Scheme". Proceedings of 1999 USENIX Annual Technical Conference: 81–92.</i>
[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 eDelivery profile.

4. SECURITY

4.1. Introduction

The Domibus Access Point 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 eDelivery AS4 profile. The configuration is handled via WS-Policy files and PMode configuration. All webservice security is enforced by the Apache CXF/WSS4J/Santuario frameworks [REF5].

4.3.1. Certificate Configuration

The location and credentials of private and public certificates used by CXF are configured in the **domibus.properties** property 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 server’s 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. Domibus Admin console

Access to the Domibus Administration Console is secured with username/password. The user credentials are stored in the database, and they are managed by an authentication provider that uses a Bcrypt strong hashing function for encoding them.

Domibus also supports the EU Login authentication provider which uses an LDAP based mechanism.

Currently it is not possible to add other authentication providers without modifying Domibus itself, so this feature is not opened to users.

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.

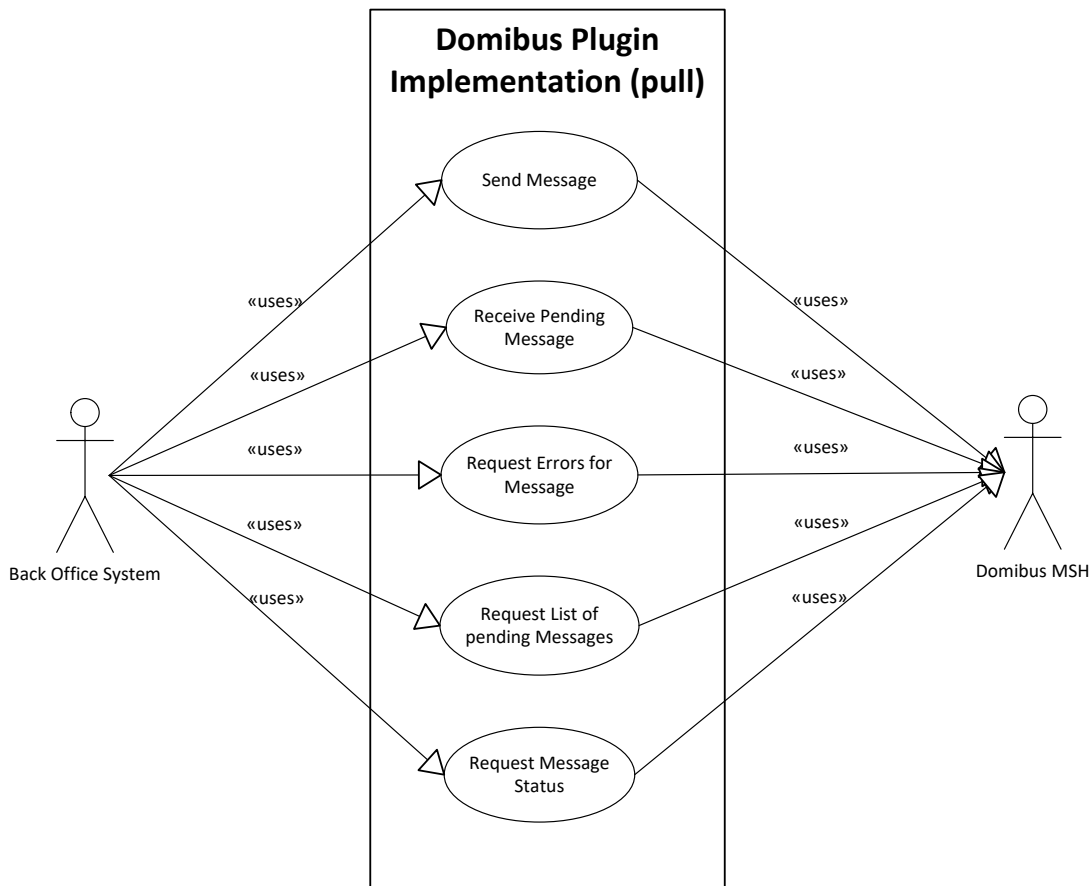
5.1. Selection Rationale

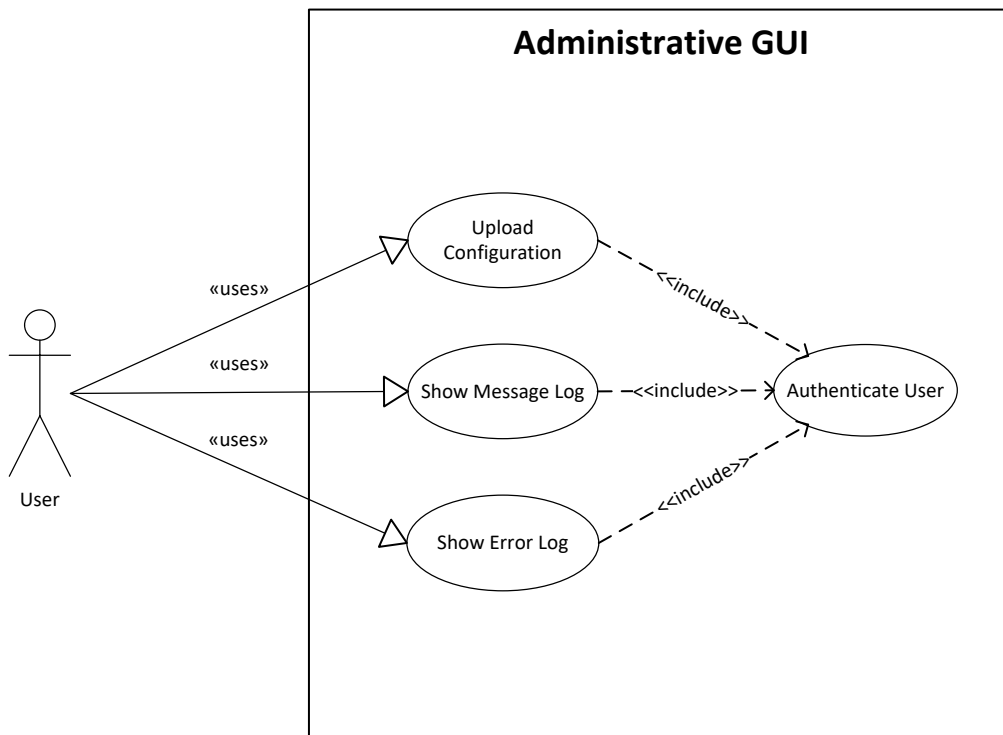
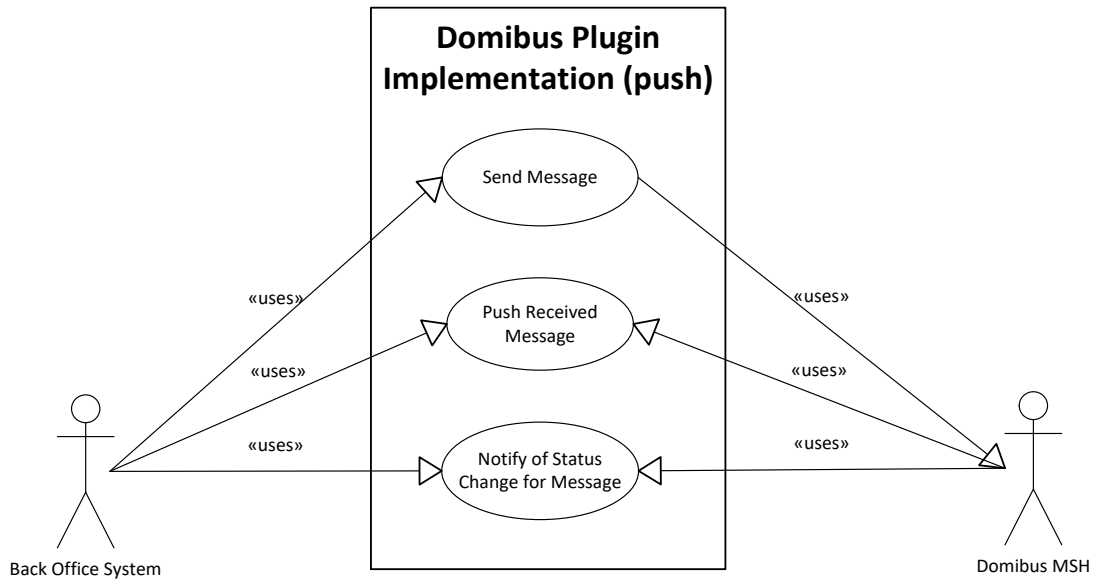
The use cases relevant for the architecture have been selected based on the following criteria:

- 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





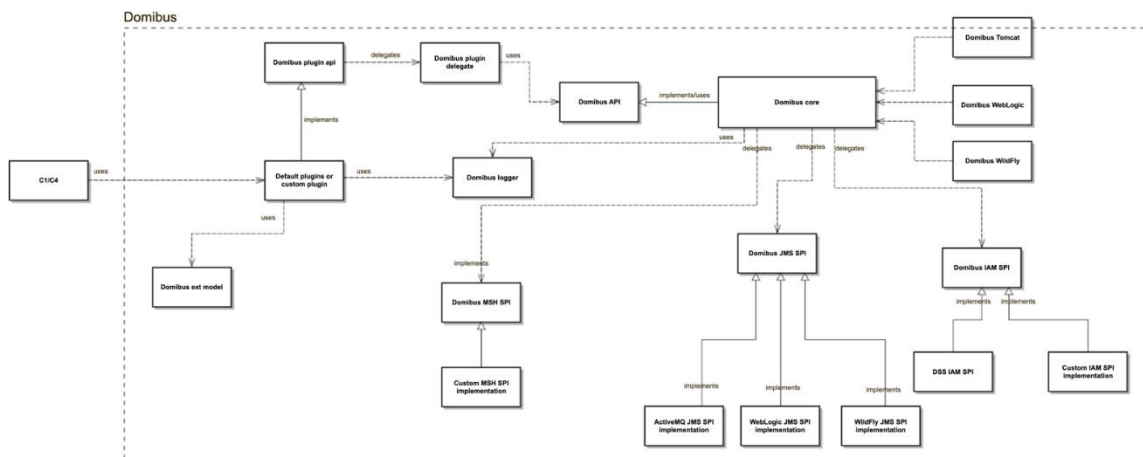
6. LOGICAL VIEW

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 Components

The following diagram provides a high-level view of the main components of the system.



You can find below a short description for each component:

- C1/C4 – the backend system which integrates with Domibus
- Domibus plugin API – the API that is used to implement a plugin
- Default plugins or custom plugin – the Domibus default plugins (e.g.: WS Plugin, JMS Plugin and the FS Plugin) or a custom plugin
- Domibus ext model – domain model used by Domibus plugins
- Domibus plugin delegate – delegates calls from the plugins to the Domibus core
- Domibus logger – Domibus custom logger used by Domibus plugins and Domibus core
- Domibus MSH SPI – the API used to implement a Domibus extension
- Domibus API – domain and services used by the internal Domibus modules
- Domibus core – the core implementation Domibus
- Domibus Tomcat – implementation and configuration specific to the Tomcat server
- Domibus WebLogic - implementation and configuration specific to the WebLogic server

- Domibus WildFly - implementation and configuration specific to the WildFly server
- Domibus JMS SPI – the internal API used to support a specific JMS broker implementation
- ActiveMQ JMS SPI – implementation to support the ActiveMQ classic JMS broker
- WebLogic JMS SPI – implementation to support the WebLogic internal JMS broker
- WildFly JMS SPI - implementation to support the WildFly internal JMS broker
- Domibus IAM SPI – the API used to implement an IAM extension
- DSS IAM SPI – the implementation of the IAM extension using the DSS library

6.2.1. Back office system (Corner 1/4)

The purpose of Domibus as an Access Point 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 taken 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 three default plugins, which can be used to integrate a back-office, acting as C1 or C4, with Domibus: Web Service Plugin, File System Plugin and JMS Plugin.

A custom plugin can be implemented in case one of the default plugins do not accommodate the needs of a back-office.

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 eDelivery profile is done inside this package. It can be deployed on Tomcat, WebLogic and WildFly.

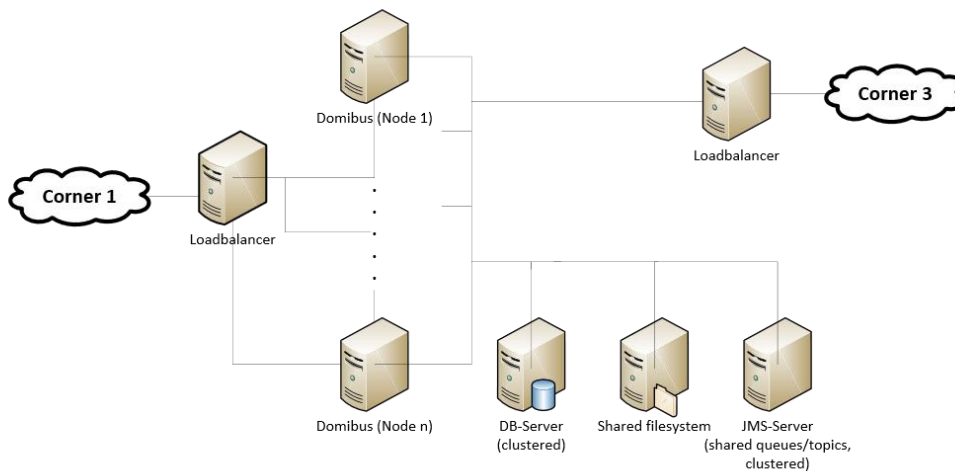
6.2.6. Administrative GUI

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

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 or Oracle) 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.

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 is exposed on the internet that accepts the AS4 requests, and it is the one that is called by the external systems. The **AS4 Message Dispatch Service**, used by Domibus acting as C2, is a web service client capable of sending AS4 requests and is responsible for sending messages to other C3 Access Points . 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.

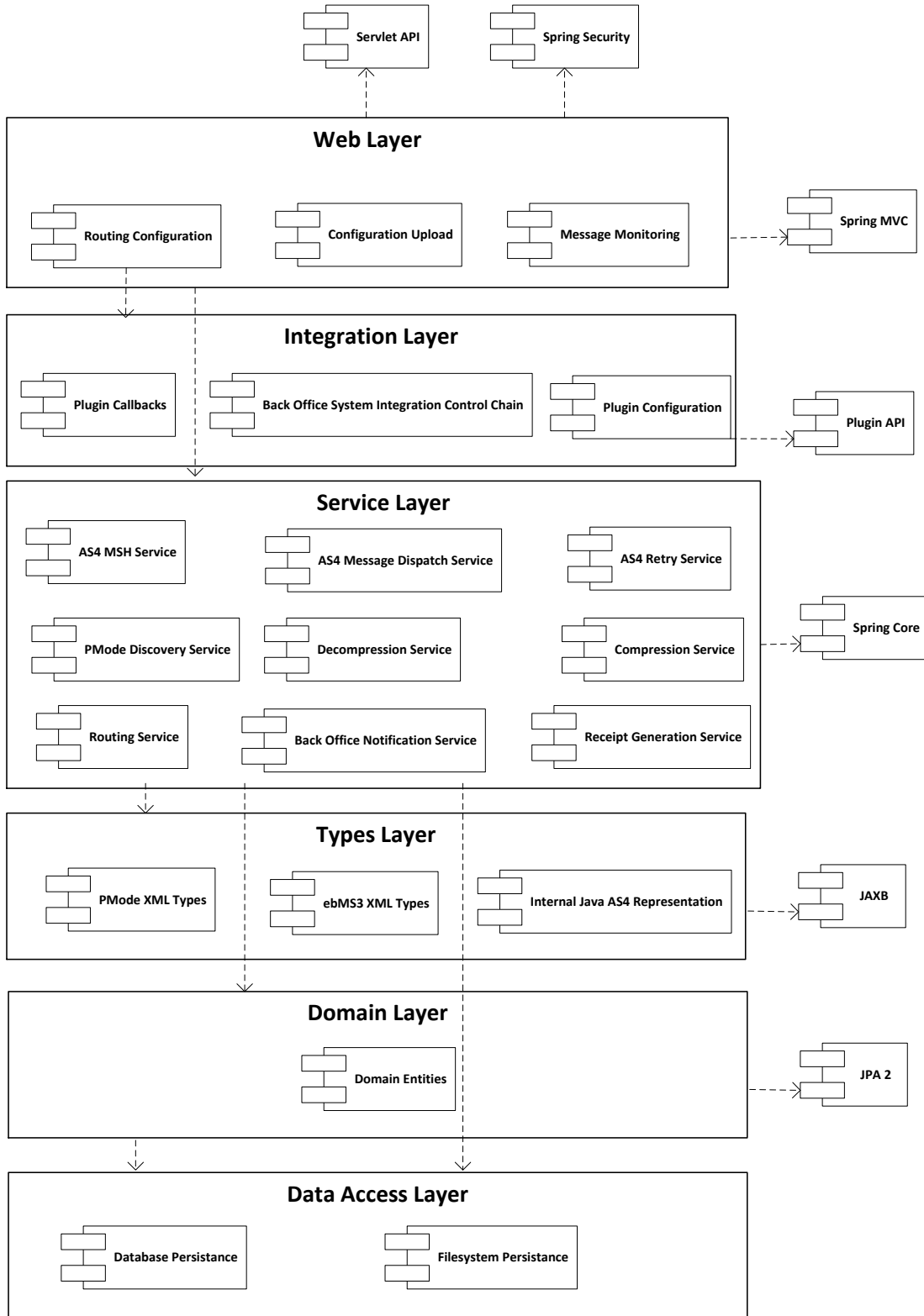
The **Services Layer** offers access to the domain objects of the platform as well as to the platform data layer. These services are Plain Old Java Objects relying on the Spring framework for dependency injection and for transaction management.

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.

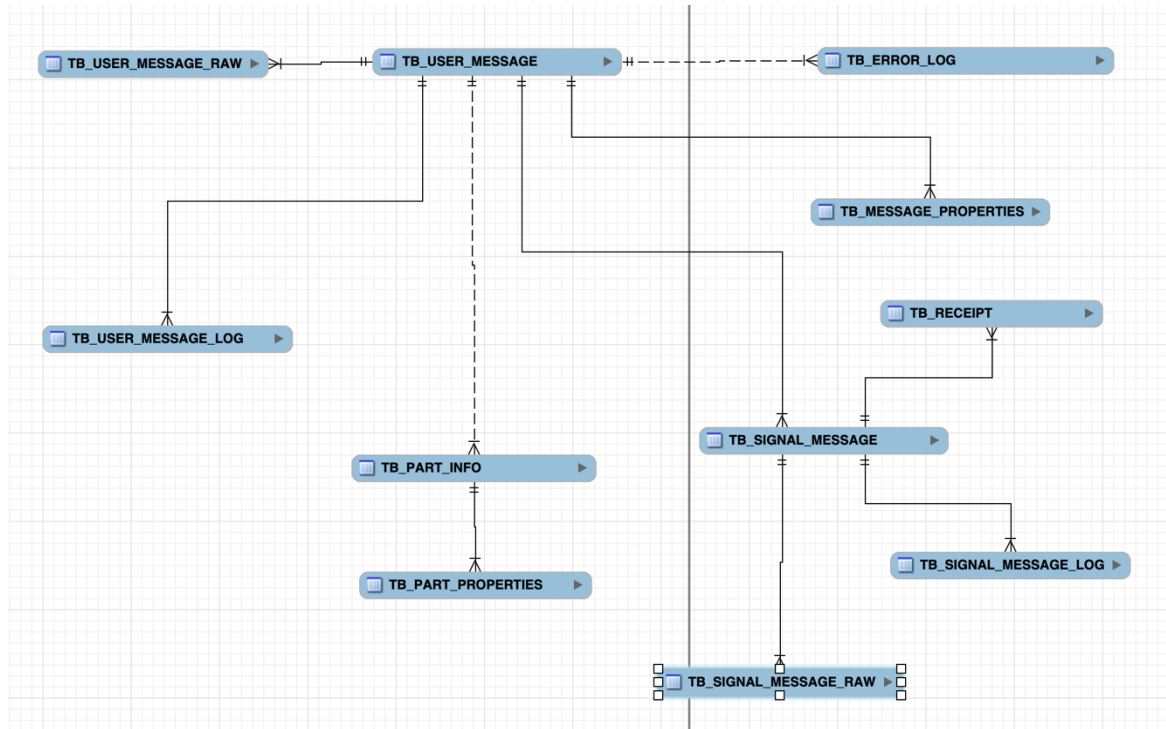
All these layers run on a Java Servlet Container.



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:



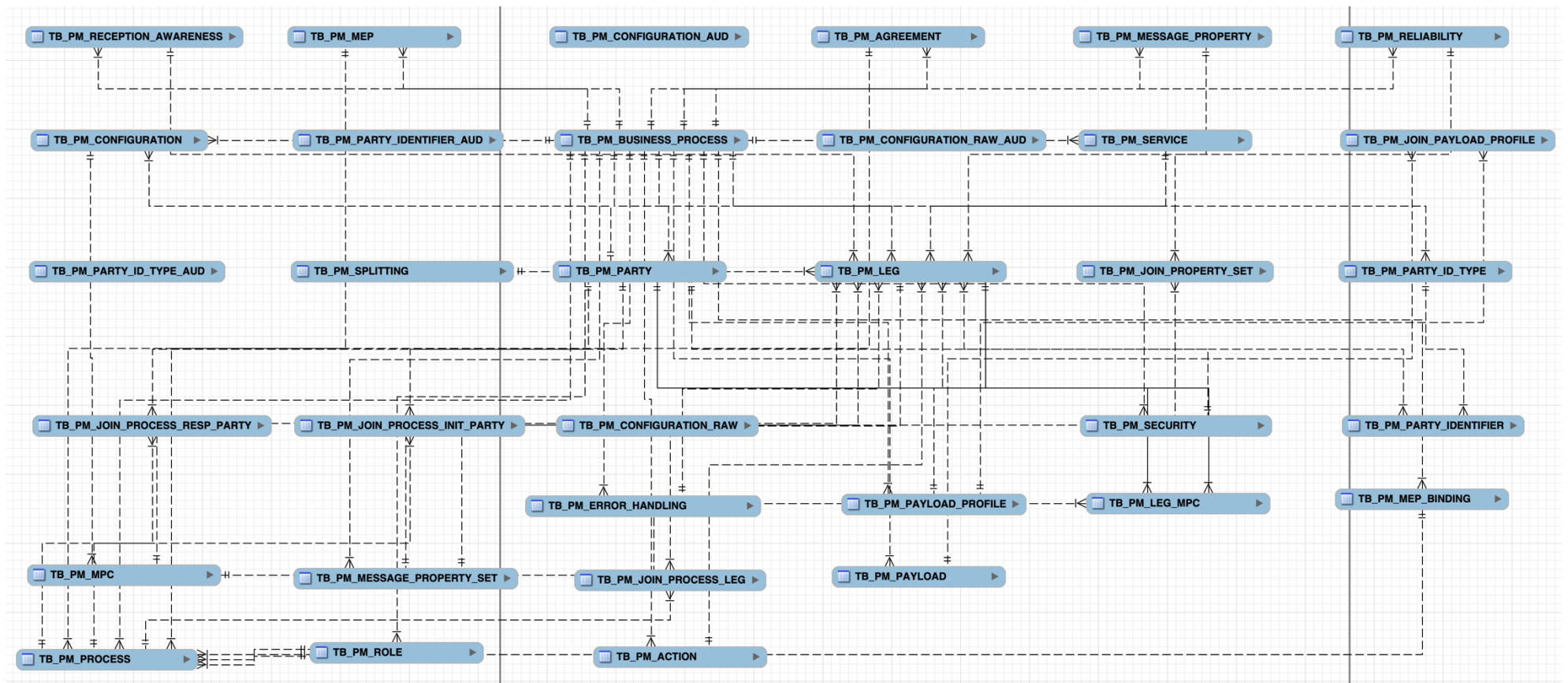
The above tables represent a mapping of the ebMS3 XSD to database tables.

The TB_USER_MESSAGE table contains information about the AS4 UserMessage metadata (both sent and received ones).

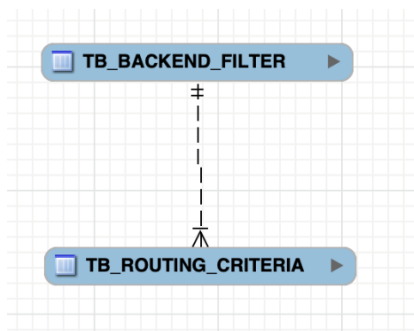
The TB_USER_MESSAGE_LOG table contains runtime information about the User Message (both sent and received ones) such the message status, retry information, etc

The TB_USER_SIGNAL_MESSAGE table contains information about the AS4 SignalMessage metadata (both sent and received ones)

The TB_SIGNAL_MESSAGE_LOG table contains runtime information about the Signal Messages(acknowledgements) for sent User Messages

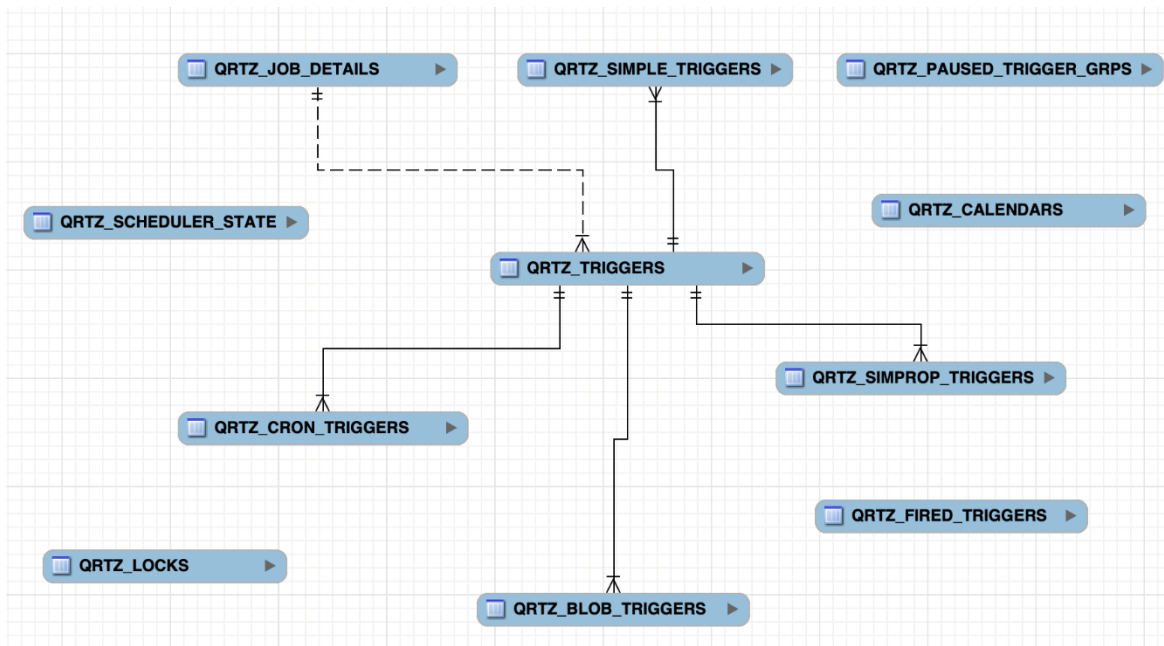


The above tables represent a 1:1 mapping of the PMode configuration XSD to database tables.



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 above tables are used by the Quartz library which is used by Domibus for crontab like jobs.

9.1.1. Data Auditing

The database tables contain information which provide audit information about their entries including the name of the user who created or last updated a particular entry or about the creation time or the last update time of a particular entry. The stored values are the following:

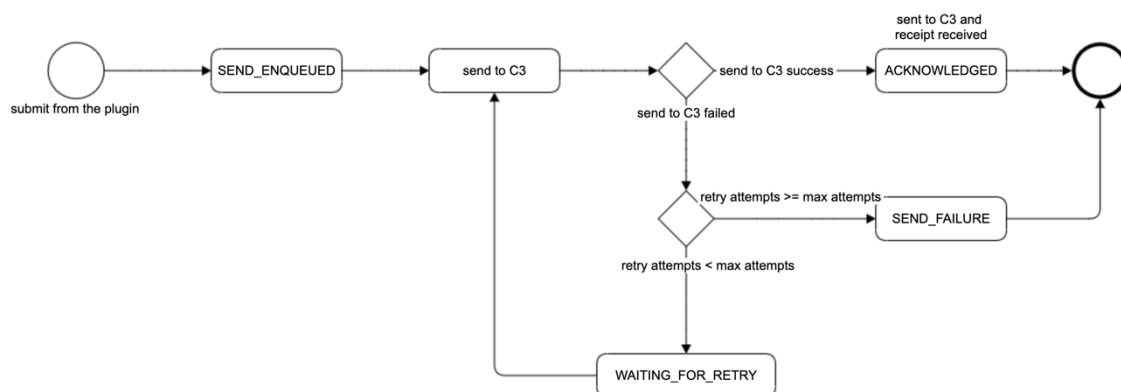
CREATED_BY, MODIFIED_BY, CREATION_TIME, MODIFICATION_TIME

These values are not visible to the Domibus user and can be only inspected at the database level. Data created or updated by the Domibus users will use their user names for the CREATED_BY and MODIFIED_BY columns. Data created using plugin users will use their plugin user names for the CREATED_BY and MODIFIED_BY columns. The rest of the entries (e.g. created during the invocation of an asynchronous JMS listener) will use the user name used to connect to the database for MySQL or the Oracle user schema for the CREATED_BY and MODIFIED_BY columns. There is one exception involving reference data (i.e., data that has to exist before the application can be used) on MySQL where we use the value 'DOMIBUS' for the CREATED_BY and MODIFIED_BY columns.

9.2. State Machines

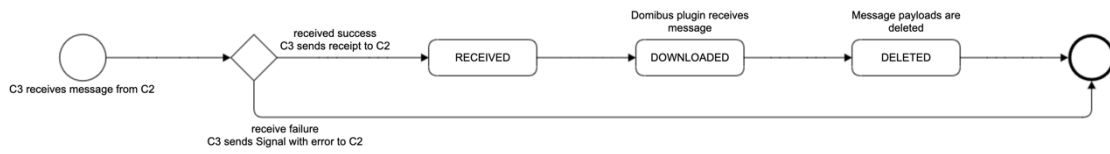
9.2.1. Outgoing Message State Machine

The outgoing messages have the following state machine:

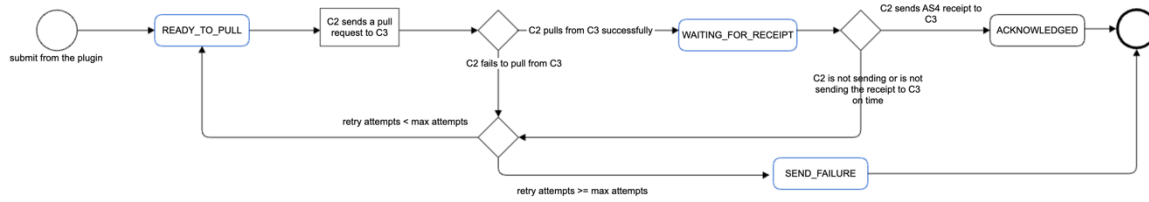


9.2.2. Incoming Message State Machine

The incoming messages have the following state machine:



9.2.3. Message State Machine for pull messages



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 eDelivery 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 Access Point.

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

11. QUALITY

The architecture of Domibus contributes to improve extensibility, reliability and portability.

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, WebLogic and WildFly 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 additional external systems need to be supported by Domibus.

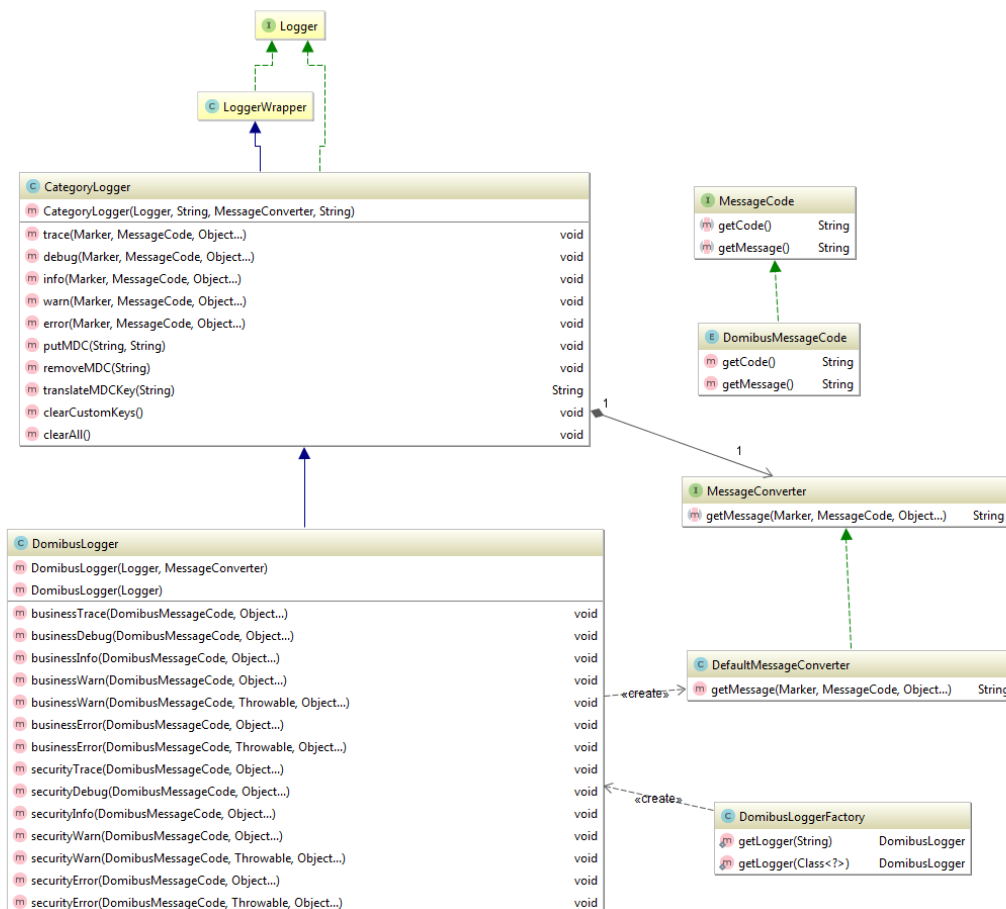
The usage of JPA to access the database makes it easy for implementers to change the relational database used to store the platform data.

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.

- `error.log`: This log file contains all the errors which occurred in Domibus including errors from third party libraries used by Domibus.

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 identify 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 [{}]
SECURITY	SEC-010	No security policy (intended for testing alone) is used. Security certificate validations will be bypassed!
SECURITY	SEC-011	User [{}] is trying to access a message having final recipient: [{}]
SECURITY	SEC-012	X509Certificate invalid or not found
SECURITY	SEC-013	The user [{}] is unknown
SECURITY	SEC-014	The user [{}] is not active
SECURITY	SEC-015	The user [{}] is suspended
SECURITY	SEC-016	The user [{}] is trying to login with bad credentials
SECURITY	SEC-017	The user [{}] is locked after trying to login for [{}] wrong attempts.
SECURITY	SEC-018	The certificate with alias [{}] will be revoked on [{}]
SECURITY	SEC-019	The certificate with alias [{}] is revoked since [{}]
SECURITY	SEC-020	The password for user [{}] will expire on [{}]
SECURITY	SEC-021	The password for user [{}] expired on [{}]

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	Matching Process or Leg not found for agreement [{}], senderParty [{}], receiverParty [{}], service [{}] and action [{}]. Process mismatch details:[{}]. Leg mismatch details:[{}].
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
BUSINESS	BUS-057	No Role with value [{}] has been found
BUSINESS	BUS-058	Party with name [{}] has not been found
BUSINESS	BUS-059	Message with id [{}] has not been found
BUSINESS	BUS-060	Message with id [{}] has been consumed from the queue [{}]
BUSINESS	BUS-061	Received payload with cid [{}] for message [{}] of size [{}] (in bytes)
BUSINESS	BUS-062	Saved payload with cid [{}] for message [{}] of size [{}] (in bytes) for sending
BUSINESS	BUS-063	Notifying about message status change from [{}] to [{}]
BUSINESS	BUS-064	Message submitted
BUSINESS	BUS-065	Message submission failed
BUSINESS	BUS-066	Message retrieved
BUSINESS	BUS-067	Message retrieval failed
BUSINESS	BUS-068	Test message successfully received from [{}] to [{}]

BUSINESS	BUS-069	Failed to receive test message from [{}] to [{}]
BUSINESS	BUS-070	Preparing to send test message from [{}] to [{}]
BUSINESS	BUS-071	Test message sent successfully from [{}] to [{}]
BUSINESS	BUS-072	Test message sending from [{}] to [{}] failed
BUSINESS	BUS-073	Message property [{}] exceeds [{}] characters limit
BUSINESS	BUS-074	Receiver Party id [{}] found for value [{}]
BUSINESS	BUS-075	Receiver Party id not found for value [{}]
BUSINESS	BUS-076	Duplicate Message Property found for property name [{}]
BUSINESS	BUS-077	Payload size is greater than maximum size [{}] defined in payload profile [{}]
BUSINESS	BUS-078	Mandatory Message Header metadata [{}] is not provided
BUSINESS	BUS-079	Value of [{}] is too long (over 255 characters). Value provided: [{}]
BUSINESS	BUS-080	Value of [{}] does not conform to the required MessageIdPattern: [{}]. Value provided: [{}]
BUSINESS	BUS-081	Message with id [{}] already exists. Message identifiers must be unique

12.3. Logging and multitenancy

In order to associate each log statement to a specific domain the **Logback** format pattern contains the domain name.

Eg: `<pattern>%d{ISO8601} [%X{d_user}] [%X{d_domain}] [%X{d_messageId}] %5p %c{1}:%L - %m%n</pattern>`

The domain name is added in the MDC context so that every log statement contains the domain name.

The domain name is added in the MDC context as soon as a thread is started:

- For a web service, a CXF interceptor is adding the domain to MDC as close as possible to the START phase. The MDC context value is cleared with a CXF interceptor added at before the END phase.
- For a JMS message listener, the domain name is extracted from the JMS message that is being consumed and added programmatically to MDC.

The separation of logs per domain is achieved using the existing Logback marker mechanism and a Logback configuration file distributed in each server configuration archive. As a result, separate log files are created containing only the logs for one domain.

For instance, for a domain named DOMAIN1 the following log files are saved under the logs/DOMAIN1 directory: DOMAIN1-domibus.log, DOMAIN1-business.log, DOMAIN1-security.log provided that the DOMAIN1-logback.xml is configured.

To separate the configuration of each logger instance per domain, the following provisions have been made:

- The loggers defined in the domain-specific logback xml file have names prefixed with *\${domainName}*. (eg. `<logger name="${domainName}.eu.domibus" level="INFO">`);
- Each logger in the application will have as many instances as there are domains, each instance having the configuration for a particular domain with all instances being stored in *DomibusLoggersCache*. The appropriate logger instance to perform an operation is selected using a proxy mechanism each time when a logging method is invoked.

This configuration is managed in the Domibus **logback.xml** file and it is independent of the Domibus application.

Note that this mechanism applies only to eu.domibus loggers. Loggers from third-party libraries cannot be configured independently for a tenant, they should be added to the main logback.xml file with the settings that apply to all tenants.

For more details on how to configure the Domibus logging, see the Domibus Administration Guide.

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”

Domibus has two types of caching mechanisms available: local and distributed.

The local cache is available when Domibus is deployed in a single instance and also in a cluster. The local cache must be used when you don't want to replicate the cache amongst the cluster deployment.

The distributed cache is only available in a cluster deployment. It should be used when you want to replicate the cache amongst the cluster members. The cache replication is performed automatically.

13.1. Local cache

Domibus uses Ehcache implementation for local caching. Ehcache is configured in `${domibus.config.location}/internal/ehcache.xml`. All the caches defined in the ehcache.xml file are preconfigured with default values and are commented out. In order to customize one of the cache configuration you can uncomment it and adapt the necessary values.

The local cache is also available to the plugins. A plugin can define its own cache configuration on top of the existing configuration provided in Domibus. For more details how to configure the local caching at plugin level, please check for more details in the Plugin Cookbook

13.2. Distributed cache

In a cluster deployment, Domibus activates also the distributed cache. The distributed cache is implemented with the Hazelcast library. The list of the Hazelcast cluster members are specified in `domibus.properties` and a Hazelcast member is created on each cluster machine.

In a cluster deployment, once you add an entry in a distributed cache, the change is replicated automatically amongst the cluster members. By default, a distributed cache is configured with a near cache in order to reduce the network overhead between the cluster members.

The distributed cache is accessible via Java API and via REST.

In a non-cluster deployment, the distributed cache defaults to the local cache.

The distributed cache is also available to the plugins. A plugin can define its own cache configuration using the API from the `domibus-plugin-api` module. For more details how to configure the distributed caching at plugin level, please check Domibus Plugin Cookbook document.

14. MULTITENANCY

14.1. General

There were multiple options to choose from to support Multitenancy:

1. Selected option: **One Schema per tenant**: tenant's data is saved in the same database for all tenants but in different schemas. When a new tenant needs to be added a new related DB schema is created in the same database instance. It is easier to add new tenants comparing with the **DB per tenant** approach, as the same connection pool can be reused. Switching between tenants is performed centrally by selecting the DB schema related to the tenant. A huge advantage of this approach is that the application code impact is limited compared to the **Discriminator field** approach described below.
2. **One DB per tenant**: each tenant has its own separate database. This is the highest level of isolation, however it is complex and cumbersome to maintain. Whenever a new tenant has to be added a new database instance needs to be created, a new database connection pool also needs to be created in Domibus which points to the tenant database, etc...
3. **Discriminator field**: All tenants' data is saved on common tables, and each table holds a discriminator field to distinguish data from each tenant. This approach has quite some disadvantages: no physical isolation of data between tenants (a bug in the application might leak between tenants), performance decrease as the data for all the tenants are saved into the same tables (resulting in bigger tables and more complex/heavier queries), and significant changes to the application code to take into account that discriminator field.

The solution that has been chosen to implement Multitenancy in Domibus is "One **Schema per tenant**" due its many advantages. Hibernate library which is used in Domibus, comes by default with support for "One **Schema per tenant**" strategy, more info about Multitenancy in Hibernate https://docs.jboss.org/hibernate/orm/5.0/userguide/html_single/chapters/multitenancy/MultiTenancy.html .

In the following document the term *tenant* is (the technical) synonym with the (more business-oriented) term *domain*.

14.2. Identifying the domain (tenant)

For every outgoing/incoming message, the related unique domain needs to be identified in order to use the configuration related to the appropriate domain (DB schema, PMode, keystore, truststore, Domibus properties, etc.).

For outgoing messages, sent by C2 to C3, the association to a specific domain is performed based on the Spring Security info available in the current thread after the authentication has been done by the plugins.

For incoming messages, received by C3 from C2, the association to a domain is based on an HTTP parameter ("domain") appended by C2 to the MSH endpoint of C3. In case the domain name sent by C2 is not defined in C3, an EBMS3 exception will be sent to C2.

Example:

Let us suppose that C3 exposes the MSH endpoint with URL:

<http://localhost:8080/domibus/service/msh>. Then C2 belonging to the domain **DIGIT** will call the MSH C3 endpoint using <http://localhost:8080/domibus/service/msh?domain=DIGIT>.

Please note that adding the HTTP parameter in the MSH endpoint is in-line with eDelivery AS4 specification.

14.2.1. Selecting the database schema

In Multitenancy mode, the database schema has to be configured per domain in the Domibus domain properties. More information on how to configure it please refer to the **Domibus Administration Console**.

14.3. User association to a domain

When a user authenticates in the Admin Console, the domain is not yet identified and Domibus must find out which DB schema to select. In order to achieve this, a **general DB schema** is used. In this general DB schema, a table tells which user that has access to the Admin Console (defined in the **tb_user** table) and to the domain he belongs to. This association is automatically updated by Domibus when users are added or removed. Therefore, a constraint has been added in Domibus: a username needs to be unique amongst the existing domains. The same mechanism and constraints apply to the **tb_user** table and has been implemented for the table supporting plugins security: **tb_authentication_entry**.

14.4. UI**14.4.1. Managing multiple domains from the Domibus Administration Console**

All the screens from **Domibus Administration Console** have been adapted in order to support Multitenancy. This paragraph briefly summarizes these changes throughout the application.

The **Login page** does not require filling in the domain. Once the user is authenticated, Domibus automatically identifies its associated domain based on the association of the users to their domain contained in the general schema. For more information how this mechanism works please check **14.2 Identifying the domain (tenant)** section.

There is a new **super** admin user for the Administration Console that has access to all domains' data but only one domain at a time. This solution has been chosen in order to simplify the user interface. In addition, the **super** admin user is responsible to manage **admin** users for every domain and is the only one authorized to do so.

14.4.2. Security

The UI is communicating with the Domibus backend via REST resources.

The UI REST resources are protected so that users belonging to domain **A** are forbidden to modify the configuration belonging to any other domain. This is implemented with an interceptor acting on the all the UI REST resources that automatically identify the DB schema associated to the logged in user (please check the **14.2.1 Selecting the database schema** section for more details). Using this strategy, a user belonging to domain **A** is not able to access the data of any other domain.

14.5. Plugins

The introduction of Multitenancy has an impact on how the plugins manage the incoming/outgoing messages.

For outgoing messages, C1 to C2, the plugins need to authenticate first so that Domibus can identify the domain of the user and treat the message accordingly.

Domibus identifies the associated domain of the user based on the Spring Security information from the current thread (for instance the logged in user id) and the users configuration in table **tb_authentication_entry**. As a result, it is mandatory for C1 to authenticate itself so that Domibus is able to determine the domain related to the authenticated user. It is possible for the same C1 to send messages to different domains C1 needs to authenticate with different user credentials

For incoming messages, e.g., from C3 to C4, the plugins have to segregate the messages based on the domain name received from Domibus and deliver to C4 only the messages associated to the C4's domain.

The changes implemented in the Default Plugin for Multitenancy are described in the following sections.

14.5.1. Security

The plugins security configuration is stored in database table **tb_authentication_entry**. As every domain has its own separate schema, the table **tb_authentication_entry** will contain entries specific to each domain.

ID_PK	CERTIFICATE_ID	USERNAME	PASSWD	AUTH_ROLES	ORIGINAL_USER	BACKEND
1	NULL	admin	\$2a\$10\$HApapHvDSiTEwijnMCvXuqUKVyyeX...	ROLE_ADMIN	NULL	NULL
2	NULL	user	\$2a\$10\$HApapHvDSiTEwijnMCvXuqUKVyyeX...	ROLE_USER	urn:oasis:names:tc:ebcore:partyid-type:unregist...	NULL
3	CN=blue_gw,O=Delivery,C=BE:103700358308...	NULL	NULL	ROLE_ADMIN	NULL	NULL

Example of the data in **tb_authentication_entry**

As mentioned in section **14.2 Identifying the domain (tenant)** the **username** should be unique amongst all domains. Moreover, there is a table **tb_user_domain** in the general DB schema that maps all usernames defined in the **tb_authentication_entry** to one associated domain.

When multiple domains are configured in Domibus, the plugins security activates automatically overriding the value configured using the following property.


```
#To activate security set this to false  
domibus.auth.unsecureLoginAllowed=false
```

If Domibus is running only with one domain, the plugins security activation is optional.

14.5.2. Plugin API

As every domain has its own dedicate DB schema, there are little changes required in the **Plugin API**. The only change that is required is to include in the class **eu.domibus.plugin.Submission** a new field named **domain**. This way the plugins can select the domain for a specific message. This is specifically useful for the incoming messages, C3 to C4, when the plugins need to segregate messages and expose to C4 only messages that are intended to C4 domain.

14.5.3. WS Plugin

Security is already implemented in the WS Plugin using the **CustomAuthenticationInterceptor** and the **eu.domibus.ext.services.AuthenticationService**, which retrieves information from the DB table **tb_authentication_entry**. For the WS Plugin security, activation is mandatory in order to use Domibus with multiple domains.

The implementation of the WS Plugin has been changed to take into account the domain according with the general requirements stated in the **14.5 Plugins** section.

14.5.4. JMS Plugin

The JMS Plugin is implemented using 5 queues:

- One queue for outgoing messages, C1 to C2: **domibus.backend.jms.inQueue**
- One queue for incoming messages, C3 to C4: **domibus.backend.jms.outQueue**
- Three queues for reporting message statuses and errors:
domibus.backend.jms.replyQueue, **domibus.backend.jms.errorNotifyConsumer**,
domibus.backend.jms.errorNotifyProducer

C1 and C4 interact with the JMS Plugin by sending/receiving messages from queues mentioned above so the JMS Plugin does not really have control on the messages once they are put in a JMS queue.

In order to segregate the data between domains, the JMS Plugin needs to connect to queues dedicated to each domain. Therefore, every domain will have its own set of 4 queues mentioned above with the exception of the **domibus.backend.jms.inQueue**. The queue **domibus.backend.jms.inQueue** is common to all the domains but when sending message to Domibus, C1 needs to authenticate with specific domain credentials. This is needed to allow Domibus to associate the submitted message with a specific domain. The association of the JMS queues and the domain are be done in the **jms-plugin.properties** file.

In order to make the migration easier the existing queue names used by the JMS plugin and associated to the **default** domain will not be modified.

The following convention to prefix the JMS queues with the domain name must be used to associate the JMS queues to a specific domain in the **jms-plugin.properties** file:

`domain_name.domibus.backend.jms.inQueue`

where **`domain_name`** is the name of the domain.

For outgoing messages, C1 to C2, C1 sends JMS messages containing the credentials of a specific domain to the **IN** queue **`domibus.backend.jms.inQueue`**. The JMS Plugin reads the JMS message, performs the authentication using the credentials sent by C1 and determines the domain based on Spring Security information from the current thread.

For incoming messages, C3 to C4, the JMS Plugin receives the domain name from Domibus API and then sends the incoming message to the JMS **OUT** queue associated to the domain where C4 is listening to.

14.5.5. FS Plugin

The FS Plugin has been designed in such a way that it is already domain aware. Briefly, the domain concept is implemented in the FS Plugin as follows:

- A file system location is defined per domain, which is protected with username/password. The username/password credentials are defined per domain in the FS Plugin property file
- In order to send messages, a user α belonging to domain A will copy the payloads to be "sent" directory in the file system location configured for domain A (the user α must have access to the protected file location). A similar process is happening when user A wants to retrieve messages.

For outgoing messages (C1 to C2), the FS Plugin authenticates itself using credentials (username/password) configured per FS Plugin domain. These new credentials are configured in the FS Plugin properties file. Once the user is authenticated, user information is extracted from the Spring Security data, associated to the current thread and passed to the Domibus Core.

For incoming messages (C3 to C4), the FS Plugin receives the domain name from Domibus so that the incoming messages are into the directory associated to the respective domain.

The domains configured in Domibus, and the domains configured in the FS Plugin properties must match. An error will be raised if the domain is not configured as needed in the FS Plugin properties.

14.6. Domibus Properties

The Domibus properties defined in the **`domibus.properties`** file are used for the **single** domain, when Domibus manages one single domain.

When Domibus is configured with multiple domains, several properties will have to be customized per domain. More information on which properties can be overridden per domain are available in the **Domibus Administration Guide**.

There are 3 types of properties in multitenancy mode:

1. Global/infrastructure - properties that have a meaning for the whole application and, as such, a single value

2. Domain - properties that can/should be customized for each domain. In case a value is not defined for a domain, it falls back to the one defined in global properties file (or an error is thrown in case the fallback is not allowed by the property metadata)
3. Super – properties applicable to super-users. Same behaviour as domain properties.

To distinguish between these types of properties, one needs to define a property **metadata** in the appropriate **metadata manager** (each module has its own property metadata manager). If a property does not have a metadata defined, then it is treated as global.

In order to define a property, the following conventions are used:

Property Type	File where it is defined	Property name
Global/Infrastructure	domibus.properties	domibus.config.location
Domain property, domain1	domain1-domibus.properties	domain1.domibus.security.keystore.location
Domain property, default domain	default-domibus.properties	default.domibus.security.keystore.location
Super property	super-domibus.properties	super.domibus.console.login.maximum.attempt

All common properties are defined in **core metadata manager**. The properties that are specific to a server (Tomcat, WildFly, WebLogic and WebLogic-ecas) are defined in a **specific metadata manager** like TomcatMetadataManager. All these properties are managed by the Domibus Property Provider.

These are considered as **internal properties**, as opposed to properties defined in **external modules** and **plugins** (DSS module, JmsPlugin). Each external module has also a **property manager** that gets and sets its own properties.

External modules can manage their properties by using their own property bag (as do default WS and FS plugins) or by delegating to the Domibus Property Provider bag (as do default JMS plugin and DSS module).

All property metadata, both internal and external, are managed centrally by the Domibus Property Provider.

The property metadata is used by the Domibus Property Provider to determine how and from where to get and set its value.

Domibus Property Provider is the single-entry point for getting and setting property values in MSH.

An external module can use its own property manager to do the same or it can call the Domibus Property Provider Delegate to do the same (as it manages all properties and knows how to route the call to the appropriate module manager).

14.7. Message Payloads

Domibus supports two strategies for saving the messages payloads: in the database or in a local directory on the file disk. Each domain can customize the strategy for saving the payloads via the **domibus.properties** file.

In case a domain chooses to save the payloads in the database, the payloads segregation is ensured as only the users registered in that domain have access to the domain specific schema.

In case a domain chooses to save the payloads in a local filesystem directory configured per domain, the payloads segregation needs to be ensured via OS access rights. It is recommended that each domain configures its own dedicated filesystem directory.

14.8. Quartz

In the current version of Domibus, each domain can customize the Quartz jobs (like the retry job expression defined with the property **domibus.msh.retry.cron**). The Quartz jobs are saved in the database schemas of each domain. A **Quartz Scheduler** can only be configured to work with one DB schema at a time. In order to support Multitenancy, a **Quartz Scheduler** instance is created for each domain with specific properties for that domain. The creation of a Quartz Scheduler per domain is performed at runtime during the Domibus starts up.

15. CONTACT INFORMATION

eDelivery Support Team

By email: EC-EDELIVERY-SUPPORT@ec.europa.eu

Support Service: 8am to 6pm (Normal EC working Days)