EUROPEAN COMMISSION

DIGIT
Connecting Europe Facility

SML

Software Architecture Document

(BDMSL)

Version [2.11]

Status [Approved]

Date: 20/07/2017

Document Approver(s):

Approver Name	Role
Adrien FERIAL	Architect

Document Reviewers:

Reviewer Name	Role
Adrien FERIAL	Architect
Yves ADAM	Business Analyst

Summary of Changes:

Version	Date	Created by	Short Description of Changes
1.0	24/07/2015	Adrien FERIAL	Initial Version
1.1	15/10/2015	Adrien FERIAL	Changes after comments from Benoît DEBROUX, Sandro D'ORAZIO and Olivier DERVEAU
1.2	13/01/2016	Adrien FERIAL	Changes Related to ROLE_ADMIN
2.0	01/06/2016	Yves ADAM	Merge with technical design document
2.1	26/07/2016	Flavio SANTOS	Replacing the word CIPA by BDMSL
2.2	26/08/2016	Yves ADAM	Adjust some formatting errors
2.3	29/08/2016	Flavio SANTOS	WebContext for Jboss Added and DNS algorithm for NAPTR replaced by SHA-256 Base32
2.4	19/09/2016	Adrien FERIAL	Added new error code
2.5	27/09/2016	Flavio SANTOS	Data model diagram update
2.6	24/01/2017	Flavio SANTOS	Granting SMP Role to multiple domains
2.7	06/03/2017	Flavio SANTOS	Updating Migration Key Constraint
2.8	07/03/2017	Tiago MIGUEL	Changes related to Is Alive
2.9	24/04/2017	Flavio SANTOS	Added multiple domains configuration
2.10	16/06/2017	Tiago MIGUEL	Data model update regarding subdomains
2.11	18/07/2017	Flavio SANTOS	Added BlueCoat Authentication activation flag. Added regular expression configuration to validate participant id. Added logical address protocol configuration flag

Table of Contents

1. INTRODUCTION

1.1. Purpose

SML was initiated by PEPPOL [REF2] The PEPPOL SML specification was submitted as input to the OASIS BDXR TC (Business Document Exchange Technical Committee) with the intent of defining a standardized and federated document transport infrastructure for business document exchange. They resulted into a new committee specification: BDXL (Business Document Metadata Service Location)[REF3].

In WP6 [REF5] , eSENS defines the Service Location ABB based upon OASIS BDXL specification, compliant with the legacy SML specification.

The eDelivery Business Document Metadata Service Location application (BDMSL) is the sample implementation of the Service Location ABB.

This document is the Software Architecture document of the eDelivery Business Document Metadata Service Location application (BDMSL) sample implementation. It intends to provide detailed information about the project:

- An overview of the solution
- The different layers
- The principles governing its software architecture

1.2. References

#	Document	Contents outline
[REF1]	SML Specification	Defines the profiles for the discovery and management interfaces for the Business Document Exchange Network (BUSDOX) Service Metadata Locator service.
[REF2]	PEPPOL	The OpenPEPPOL Association is responsible for the governance and maintenance of the PEPPOL specifications that enable European businesses to easily deal electronically with any European public sector buyer in their procurement processes.
[REF3]	OASIS Business Document Metadata Service Location Version 1.0 (BDXL)	This specification defines service discovery methods. A method is first specified to query and retrieve a URL for metadata services. Two metadata service types are then defined. Also an auxiliary method pattern for discovering a registration service to enable access to metadata services is described. The methods

		defined here are instances of the generic pattern defined within IETF RFCs for Dynamic Delegation Discovery Services (DDDS). This specification then defines DDDS applications for metadata and metadata-registration services.
[REF4]	Implementation guidelines for the Service Location Architecture Building Block as defined by eSENS	eSENS defines a profile for the Service Location ABB based on OASIS BDXL and PEPPOL SML specification.
[REF5]	WP6 from eSENS	Work Package 6, eSENS

2. OVERVIEW OF THE SOLUTION

The eDelivery BDMSL is a solution conformant with both the SML specification and BDXL Core Implementation Conformance specification.

2.1. Service Metadata Locator (SML)

The SML is the only centrally operated component in the eDelivery Messaging Infrastructure. The dynamic discovery process begins with the establishment of the Service Metadata relating to the particular gateway to which a sender wants to transmit a message. To find the address of the Service Metadata of a participant, the Service Metadata Locator [REF1] service specification is based on the use of DNS (Domain Name System) lookups.

2.2. Business Document Metadata Service Location (BDXL)

The functionality of SML has been subsumed in a more general technical specification called the Business Document Metadata Service Location (BDXL) [REF3].This specification is based on DNS, like SML, but is based on a different type of DNS resource records called URI-enabled Naming Authority Pointer records (U-NAPTR), which are defined to support Dynamic Delegation Discovery Service (DDDS). The result of a query is a full URI, which can use HTTPS and supports server (and optionally client) authentication.

3. PRESENTATION OF THE DIFFERENT LAYERS

A multi-layered architecture requires respecting some principles:

- Structure of the java packages: in a project, every layer is represented as a package containing all the Java components of this layer
- Calls between the layers: The calls must respect the hierarchy of the layers and must be performed only by using interfaces as shown in the diagram below:

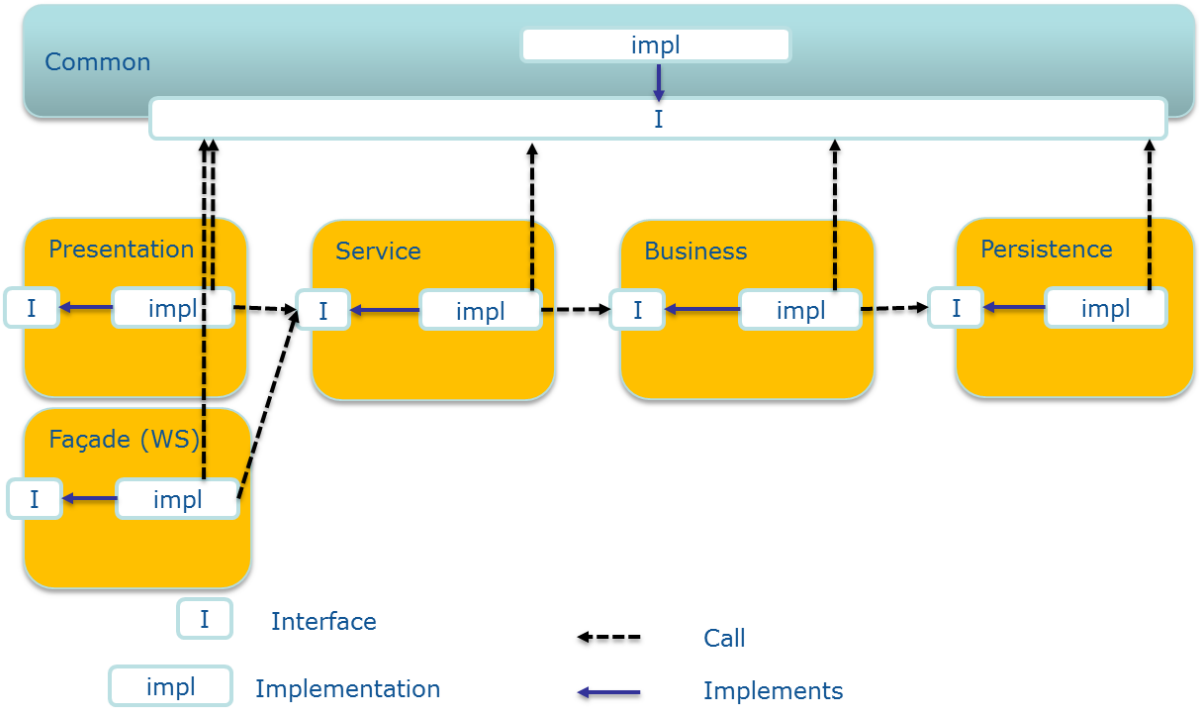

Figure 1 - Inter-layers interactions

3.1. Presentation layer

The presentation layer manages the Graphical User Interface (GUI: pages, graphical components, etc.) and the page flow.

This layer mainly handles:

- The GUI
- Interaction with the user
- The page flow
- User sessions
- Calls to the service layer through a controller

3.1.1. Naming convention

- Java package for controller: eu.europa.ec.bdmsl.presentation.controller
- Implementation: eu.europa.ec.bdmsl.presentation.controller.<PageName>Controller
- Folder for the JSP files : src/main/webapp/WEB-INF/jsp

3.1.2. Dependencies

In this layer, only the following calls are allowed:

- Calls to technical components
- Calls to the service layer
- Calls to the common package

3.1.3. Frameworks and patterns

The presentation layer relies on the Spring MVC (Model-View-Controller) framework.

The views are represented by JSP files. These files are bound to controllers. The models are built from calls to the service layer. They are then returned to the views. The presentation layer includes the controllers.

For instance, this is the `ListDNSController` implementation class:

```
package eu.europa.ec.bdmsl.presentation.controller;

[...]

@Controller
public class ListDNSController {

    @Autowired
    private ILoggingService loggingService;

    @Value("${dnsClient.enabled}")
    private String dnsEnabled;

    @Value("${dnsClient.server}")
    private String dnsServer;

    // Path to the service
    @RequestMapping("/listDNS")
    public String listDNS(Model model) {
        loggingService.debug("Calling listDNS...");
        [...]
        // We can add any object in the model and retrieve them in the views
        model.addAttribute("dnsEnabled", dnsEnabled);
        // bound to listDNS.jsp file in the src/main/webapp/WEB-INF/jsp folder
        return "listDNS";
    }
}
```

In the `listDNS.jsp` file, the model can be accessed like this:

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
    <%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
    <head>
        <title>BDMSL Service</title>
    </head>
    <body>
        <h1>ListDNS</h1>
        <c:choose>
            <!-- Access to the model -->
            <c:when test="${dnsEnabled}">
                [...]
            </c:when>
```

```
            <c:otherwise>
                <ul>
                    <li>The DNS client is disabled.</li>
                </ul>
            </c:otherwise>
        </c:choose>
    </body>
</html>
```

3.2. Service layer

The service layer is the most important layer of the application as it coordinates the calls to the business rules.

The service layer handles the transaction management. It creates the transaction and instantiates the technical objects required (sessions, connection, etc.). The transactions manage the commit/rollback depending on the errors raised by the different layers that it calls (service, business, and persistence).

The transaction management is handled by Spring through the use of the annotation `@Transactional`. Spring transparently encapsulates the calls to the different services and create if necessary transactions if the configuration requires it.

By default, the transactions are in "read-only" mode (attribute `read-only = true`) at the class level.

```
...
@Transactional(readOnly = true)
public class ManageServiceMetadataServiceImpl extends AbstractServiceImpl
implements IManageServiceMetadataService {
...
    @Transactional(readOnly = false, rollbackFor = Exception.class)
    public void create(final ServiceMetadataPublisherBO smpBO) throws
BusinessException, TechnicalException {
     ...
    }

    public ServiceMetadataPublisherValue read(ServiceMetadataPublisherBO
messagePartBO) throws BusinessException, TechnicalException {
     ...
   }
}
```

The service layer performs different calls to the service/business layers.

The objects from the service layer are POJO that implement the singleton pattern. The objects from the different layers are injected by Spring by setters.

3.2.1. Naming convention

Naming convention for:

- Package: eu.europa.ec.bdmsl.service
- Interface: eu.europa.ec.bdmsl.service.I<InterfaceName>Service
- Implementation package: eu.europa.ec.bdmsl.service.impl

- Implementation: eu.europa.ec.bdmsl.service.impl.<InterfaceName>ServiceImpl

3.2.2. Dependencies

In this layer, only the following calls are allowed:

- Calls to technical components
- Calls to the business layer
- Calls to the common package

3.2.3. Frameworks and patterns

The service layer uses these frameworks:

- Spring: transaction management, exception handling, dependency injection

3.2.4. Development of the service layer

3.2.4.1. Interface

```
public interface IManageServiceMetadataService {

  /**
   * Retrieves the Service Metadata Publisher record for the service metadata.
   * @param serviceMetadataPublisherID the unique ID
   *           of the Service Metadata Publisher for which the record is required
   * @return ServiceMetadataPublisherBO the service metadata publisher record.
   * @throws TechnicalException Technical exception.
   * @throws BusinessException Business exception.
   */
  ServiceMetadataPublisherBO read(String serviceMetadataPublisherID)
    throws TechnicalException, BusinessException;
  ...
}
```

3.2.4.2. Implementation classes

The implementation classes extend the parent-class «`AbstractServiceImpl`» and implement their dedicated interface (here «`IManageServiceMetadataService`»).

```
@Transactional(readOnly = true)
public class ManageServiceMetadataServiceImpl extends AbstractServiceImpl
implements IManageServiceMetadataService {

  private IManageServiceMetadataServiceBusiness
manageServiceMetadataServiceBusiness;

  /*
   * (non-Javadoc)
   *
   * @see eu.europa.ec.bdmsl.service.IManageServiceMetadataService#read (String)
   */
  @Override
  @Transactional(readOnly = true)
  public ServiceMetadataPublisherBO read(String serviceMetadataPublisherID)
    throws TechnicalException, BusinessException {
    ServiceMetadataPublisherBO smpBO =
manageServiceMetadataServiceBusiness.read(serviceMetadataPublisherID);
```

```
    return smpValue;
  }
  ...
}
```

The parent-class «`AbstractServiceImpl`» contains all common attributes and methods of the implementation classes of the service layer (logging service, etc.).

3.2.4.3. Transaction configuration

The transaction management is managed by Spring.

The JDBC connections to the database are open by Spring if the processing of the service requires access to the database (though the call to the business layer).

The configuration of the transaction management is made by annotations. These annotations define the rollback policy when certain types of exception may be raised. Indeed, if a non-critical error is raised, it could be useful to perform a commit anyway.

The annotations for the transaction management are set in the service class implementations with `@Transactional`. This way, we scan specify which interface and methods are executed in a transactional context.

There are two types of transaction modes:

- **read-only** is used by default. It can perform read actions but can't write anything in the database.
- **read-write** is used for CUD methods (Create, Update, Delete).

Propagation attributes manage the opening of the transactions. In this project, the attribute REQUIRED is used. This attribute, which is used by default, means that the method must be executed in a transaction context. If the transaction doesn't exist at the time of the call, then it a new one is created.

Thus, only one transaction is allowed for a call to a method in the service layer. If the method calls itself other services, the transaction will be propagated.

By default, Spring performs a rollback when a runtime exception is thrown. This behaviour can be modified with the attribute `rollbackFor` by passing a list of exceptions for which a rollback will be performed. In the BDMSL component, we rollback for any type of exception (checked and runtime), so we set the following value: `rollbackFor = Exception.class`.

3.3. Web service package

This chapter describes the use of the Apache CXF framework in the web service package to expose SOAP and REST web services.

A web service is a service that can be remotely invoked by another system.

The services of the eDelivery BDMSL application are declared in Spring and implement the Singleton pattern.

In order to connect the classes exposed by CXF to the service class managed by Spring, we use an additional package that plays the role of **Façade**: `eu.europa.ec.bdmsl.ws`

The façades define the same interfaces as the services they are linked to. They have the same methods as the Java implementation classes of the services managed by Spring. The façades implement strictly the interface they reference and serve as a transition with the external systems, taking into consideration matters like database connection, transaction, security, etc. The façade also performs the conversion of the objects from JAXB to BO and vice-versa.

The reference to the underlying service is injected in the façade with Spring. For each method of the interface implemented by the façade, we invoke the same method as on the service implementation class:

```
[...]
public class ManageParticipantIdentifierWSImpl extends AbstractWSImpl implements
IManageParticipantIdentifierWS {

    @Autowired
    private IManageParticipantIdentifierService
manageParticipantIdentifierService;

    @Autowired
    private MapperFactory mapperFactory;

    [...]

    @Override
    @WebResult(name = "ParticipantIdentifierPage", targetNamespace =
"http://busdox.org/serviceMetadata/locator/1.0/", partName = "messagePart")
    @WebMethod(operationName = "List", action =
"http://busdox.org/serviceMetadata/ManageBusinessIdentifierService/1.0/
:listIn")
    public ParticipantIdentifierPageType list(@WebParam(partName =
"pageRequestType", name = "PageRequest", targetNamespace =
"http://busdox.org/serviceMetadata/locator/1.0/") PageRequestType
pageRequestType) throws NotFoundFault, InternalErrorFault, UnauthorizedFault,
BadRequestFault {
        ParticipantIdentifierPageType result = null;
        try {
            [...]
            // convert input from JAXB to BO
            PageRequestBO pageRequestBO =
mapperFactory.getMapperFacade().map(pageRequestType, PageRequestBO.class);

            // call the service layer
            ParticipantListBO resultParticipantBOList =
manageParticipantIdentifierService.list(pageRequestBO);
            loggingService.businessLog(LogEvents.BUS_PARTICIPANT_LIST,
pageRequestBO.getSmpId());

            // convert output from BO to JAXB
            result = mapperFactory.getMapperFacade().map(resultParticipantBOList,
ParticipantIdentifierPageType.class);
        } catch (Exception exc) {
            [...]
            handleException(exc);
        }
        return result;
    }
    ...
}
```

3.3.1. Dependencies

In this package, only the following calls are allowed:

- Calls to technical components
- Calls to the service layer
- Calls to the common package

3.3.2. SOAP web services

This section describes the general principles governing SOAP web services.

3.3.2.1. Naming convention

- Package: eu.europa.ec.bdmsl.ws.soap
- Interface: eu.europa.ec.bdmsl.ws.soap.I<InterfaceName>WS
- Implementation package: eu.europa.ec.bdmsl.ws.soap.impl
- Implementation: eu.europa.ec.bdmsl.ws.soap.impl.<InterfaceName>WSImpl

3.3.2.2. Exposing a web service

As from Java 5, the JSR 181, implemented in Apache CXF, allows declaring a Java class as a SOAP web service.

To declare a façade as a web service class, the use of the annotations `@WebService`, `@SOAPBinding` et `@BindingType` is required as follow:

```
@WebService(serviceName = "ManageServiceMetadataService", portName =
"ManageServiceMetadataServicePort", targetNamespace =
Constants.MANAGE_METADATA_SERVICE_NS)
@SOAPBinding(style = SOAPBinding.Style.DOCUMENT, use = SOAPBinding.Use.LITERAL)
@BindingType(javax.xml.ws.soap.SOAPBinding.SOAP11HTTP_BINDING)
public class ManageServiceMetadataWSImpl {
...
}
```

NB: these annotations are provided by the JSR 181 API and must be set at both the implementation class and interfaces level.

To avoid the exposition of some methods like getter/setters, we use the annotation `@WebMethod(exclude=true)`.

3.3.2.3. Declaration of the exposed services

The endpoint and the implementing classes are defined in the `cxf-servlet.xml` file. This is how we can expose the service `bdmslservice`:

```
<?xml version="1.0" encoding="UTF-8" ?>
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:jaxws="http://cxf.apache.org/jaxws"
       xsi:schemaLocation="
       http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.xsd
        http://cxf.apache.org/jaxws
        http://cxf.apache.org/schemas/jaxws.xsd">
```

```
    [...]

 <jaxws:endpoint
   id="bdmslService"
   implementor="eu.europa.ec.bdmsl.ws.soap.impl.BDMSLServiceWSImpl"
   address="/bdmslservice">
 </jaxws:endpoint>

</beans>
```

3.3.2.4. WSDL

The exposed web services are defined in a contract interface file named WSDL. In the eDelivery BDMSL application, we use a **WSDL-first approach**. It means that we first design the WSDL and generate Java code from the WSDL.

The classes are generated through the use of a maven plugin defined in the `pom.xml` file. To generate the classes, the following command line must be run. Among other operations, this command will automatically call the `wsdl2java` goal from the `cxf-codegen-plugin` plugin:

```
mvn package
```

For the SML compliance, the WSDL files are defined in the SML specification. In the eDelivery BDMSL application, they are:

- ManageBusinessIdentifierService-1.0.wsdl
- ManageServiceMetadataService-1.0.wsdl
- BDMSLService-1.0.wsdl

These files are located in the `src/main/webapp/WEB-INF/wsdl` directory.

3.3.2.5. Binding

The use of JAXB configuration allows customizing the generated sources like package or class names, and also allowing the definition of adapters between XML and Java types (marshalling/unmarshalling).

These binding files are stored as `xjb` files in the `src/main/webapp/WEB-INF/wsdl` directory.

3.3.2.6. Mapping JAXB objects / BO

JAXB objects are generated from the WSDL. Inside the different layers of the application, we only use Business Objects (BO). We don't directly use JAXB generated objects in the business logic because this would tightly couple the business logic to the WSDL. A schema change in a WSDL (such as for version update) typically leads to a different package structure for classes generated from that WSDL via JAXB. To mitigate this risk, we use different Java objects : the business objects (BO). For more information on the mapping of JAXB objects to BO, see the chapter ***8 Object mapping.***

3.3.2.7. Frameworks and patterns

- Apache CXF: Exposing SOAP/REST web services. Only in the web module service.
- Spring: dependency injection

3.3.3. Services specifications

This paragraph provides implementation details of the BDMSL.

There are 3 interfaces described in this paragraph :

Interface	Description
ManageServiceMetadataService-1.0.wsdl	Defined in the PEPPOL SML specification [REF1], in Appendix B: WSDLs.
ManageBusinessIdentifierService-1.0.wsdl	Defined in the PEPPOL SML specification [REF1], in Appendix B: WSDLs.
BDMSLService-1.0.wsdl	Contains services not covered by any specification from OASIS or PEPPOL.

3.3.3.1. ManageService Metadata Service

3.3.3.1.1. WSDL file

- ManageServiceMetadataService-1.0.wsdl

3.3.3.1.2. Operation Create()

3.3.3.1.2.1. Pre-requisites

- The user has a valid certificate[1]
- The role associated to the certificate is ROLE_SMP
- The SMP doesn't already exists in the system

3.3.3.1.2.2. Description

Establishes a Service Metadata Publisher metadata record, containing the metadata about the Service Metadata Publisher (SMP), as outlined in the ServiceMetadataPublisherService data type.

- Input CreateServiceMetadataPublisherService: ServiceMetadataPublisherService - contains the service metadata publisher information, which includes the logical and physical addresses for the SMP (Domain name and IP address). It is assumed that the ServiceMetadataPublisherID has been assigned to the calling user out-of-bands.
- Fault: unauthorizedFault - returned if the caller is not authorized to invoke the Create operation
- Fault: badRequestFault - returned if the supplied CreateServiceMetadataPublisherService does not contain consistent data
- Fault: internalErrorFault - returned if the SML service is unable to process the request for any reason

3.3.3.1.2.3. Technical design

[1] In this document, we consider that a certificate is valid if it is not revoked and not expired

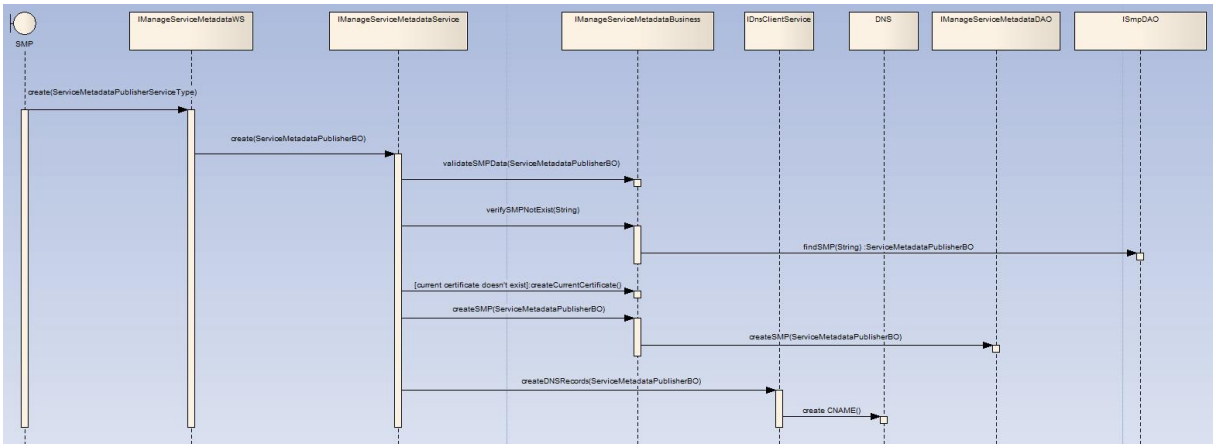

Figure 2 - Sequence Diagram - ManageServiceMetadata Create

3.3.3.1.3. Operation Read()

3.3.3.1.3.1. Pre-requisites

- The user has a valid certificate
- The role of the user is ROLE_SMP
- The SMP already exists

3.3.3.1.3.2. Description

Retrieves the Service Metadata Publisher record for the service metadata publisher.

- Input ReadServiceMetadataPublisherService: ServiceMetadataPublisherID - the unique ID of the Service Metadata Publisher for which the record is required
- Output: ServiceMetadataPublisherService - the service metadata publisher record, in the form of a ServiceMetadataPublisherService data type
- Fault: notFoundFault - returned if the identifier of the SMP could not be found
- Fault: unauthorizedFault - returned if the caller is not authorized to invoke the Read operation
- Fault: badRequestFault - returned if the supplied parameter does not contain consistent data
- Fault: internalErrorFault - returned if the SML service is unable to process the request for any reason

3.3.3.1.3.3. Technical design

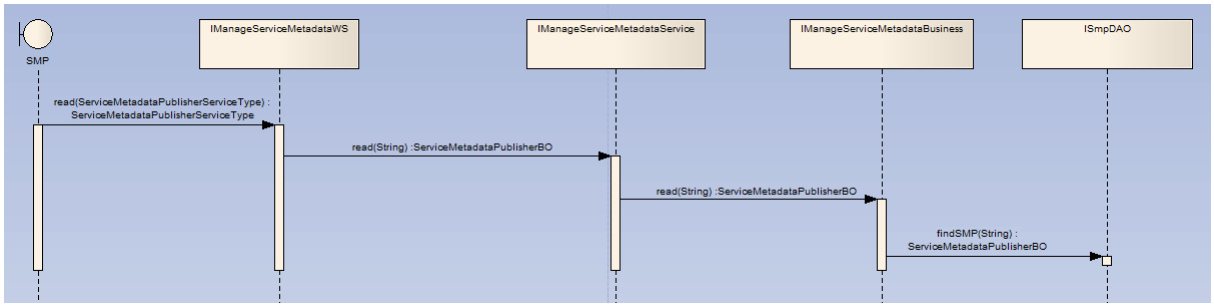

Figure 3 - Sequence Diagram - ManageServiceMetadata Read

3.3.3.1.4. Operation Update()

3.3.3.1.4.1. Pre-requisites

- The user has a valid certificate
- The role of the user is ROLE_SMP
- The SMP already exists

3.3.3.1.4.2. Description

Updates the Service Metadata Publisher record for the service metadata publisher

- Input UpdateServiceMetadataPublisheServicer: ServiceMetadataPublisherService - contains the service metadata for the service metadata publisher, which includes the logical and physical addresses for the SMP (Domain name and IP address)
- Fault: notFoundFault - returned if the identifier of the SMP could not be found
- Fault: unauthorizedFault - returned if the caller is not authorized to invoke the Update operation
- Fault: badRequestFault - returned if the supplied UpdateServiceMetadataPublisheServicer does not contain consistent data
- Fault: internalErrorFault - returned if the SML service is unable to process the request for any reason

3.3.3.1.4.3. Technical design

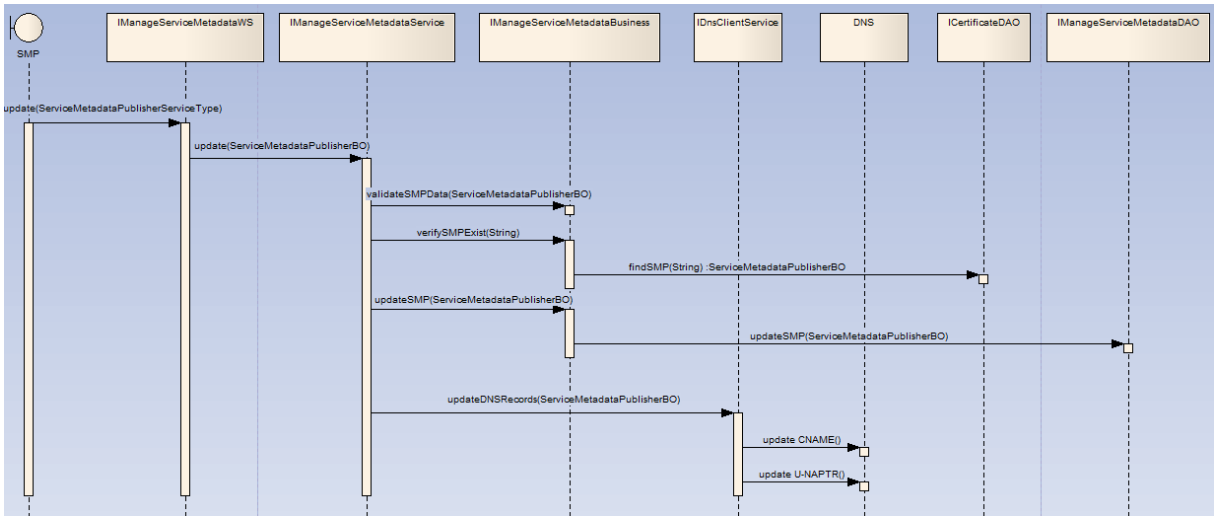

Figure 4 - Sequence Diagram - ManageServiceMetadata Update

3.3.3.1.5. Operation Delete()

3.3.3.1.5.1. Pre-requisites

- The user has a valid certificate
- The role of the user is ROLE_SMP
- The SMP already exists

3.3.3.1.5.2. Description

Deletes the Service Metadata Publisher record for the service metadata publisher.

- Input DeleteServiceMetadataPublisherService: ServiceMetadataPublisherID - the unique ID of the Service Metadata Publisher to delete
- Fault: notFoundFault - returned if the identifier of the SMP could not be found
- Fault: unauthorizedFault - returned if the caller is not authorized to invoke the Delete operation
- Fault: badRequestFault - returned if the supplied DeleteServiceMetadataPublisherService does not contain consistent data
- Fault: internalErrorFault - returned if the SML service is unable to process the request for any reason

Implementation note: If the SMP is linked to many participants, then the participants are deleted from the database and the DNS by batch of 300 elements. This is to avoid reaching the limit of the DNS protocol. Indeed, the RFC1035 of the DNS standard states that the messages are bound to 65535 bytes length.

3.3.3.1.5.3. Technical design

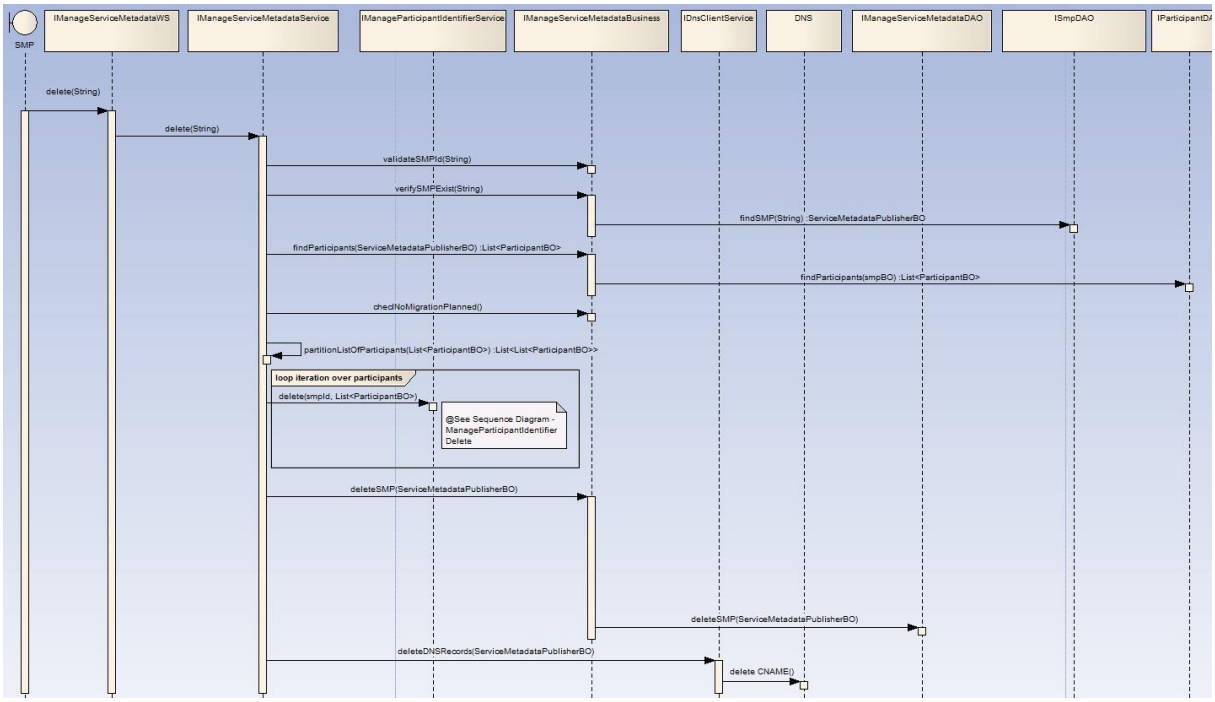

Figure 5 - Sequence Diagram - ManageServiceMetadata Delete

3.3.3.2. Manage Participant Identifier

3.3.3.2.1. WSDL file

- ManageBusinessIdentifierService-1.0.wsdl

3.3.3.2.2. Operation Create()

3.3.3.2.2.1. Pre-requisites

- The user has a valid certificate
- The SMP already exists
- The role of the user is ROLE_SMP
- The participant doesn't already exists

3.3.3.2.2.2. Description

Creates an entry in the Service Metadata Locator Service for information relating to a specific participant identifier. Regardless of the number of services a recipient exposes, only one record corresponding to the participant identifier is created in the Service Metadata Locator Service by the Service Metadata Publisher which exposes the services for that participant.

- Input CreateParticipantIdentifier: ServiceMetadataPublisherServiceForParticipantType - contains the Participant Identifier for a given participant and the identifier of the SMP which holds its data
- Fault: notFoundFault - returned if the identifier of the SMP could not be found
- Fault: unauthorizedFault - returned if the caller is not authorized to invoke the Create operation
- Fault: badRequestFault - returned if the supplied CreateParticipantIdentifier does not contain consistent data
- Fault: internalErrorFault - returned if the SML service is unable to process the request for any reason

3.3.3.2.2.3. Technical Design

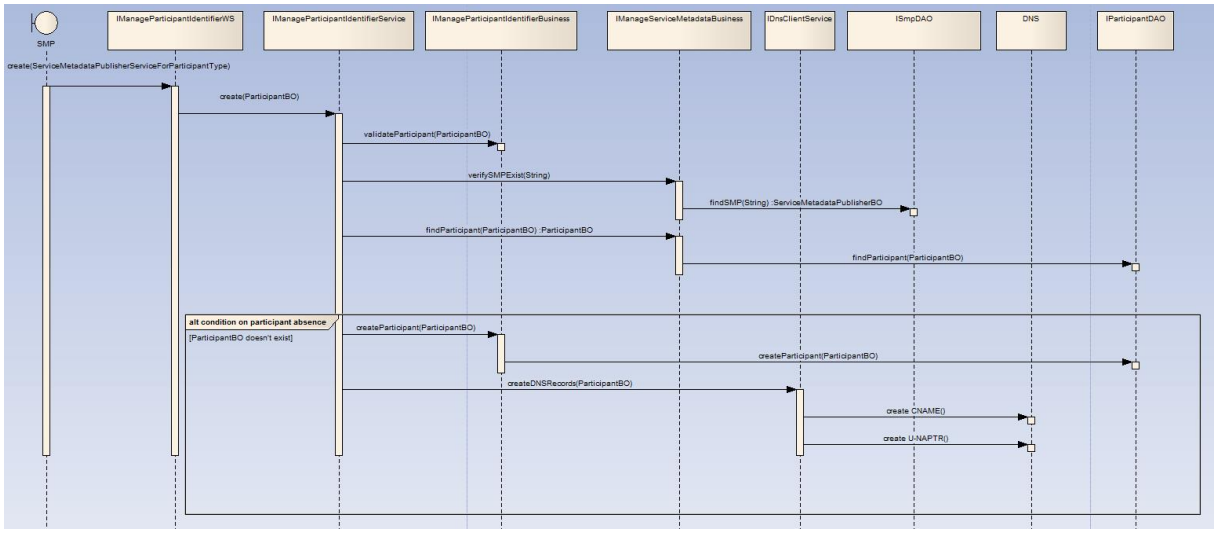

Figure 6 - Sequence Diagram - ManageParticipantIdentifier Create

3.3.3.2.3. Operation CreateList()

3.3.3.2.3.1. Pre-requisites

- The user has a valid certificate
- The SMP already exists
- The participants don't already exists

- The role of the user is ROLE_SMP
- The number of participants in the list is less than 100

3.3.3.2.3.2. Description

Creates a set of entries in the Service Metadata Locator Service for information relating to a list of participant identifiers. Regardless of the number of services a recipient exposes, only one record corresponding to each participant identifier is created in the Service Metadata Locator Service by the Service Metadata Publisher which exposes the services for that participant.

- Input CreateList: ParticipantIdentifierPage - contains the list of Participant Identifiers for the participants which are added to the Service Metadata Locator Service. The NextPageIdentifier element is absent.
- Fault: notFoundFault - returned if the identifier of the SMP could not be found
- Fault: unauthorizedFault - returned if the caller is not authorized to invoke the CreateList operation
- Fault: badRequestFault - returned if:
- The supplied CreateList does not contain consistent data
- The number of participants in the list is greater than 100
- Fault: internalErrorFault - returned if the SML service is unable to process the request for any reason

3.3.3.2.3.3. Technical Design

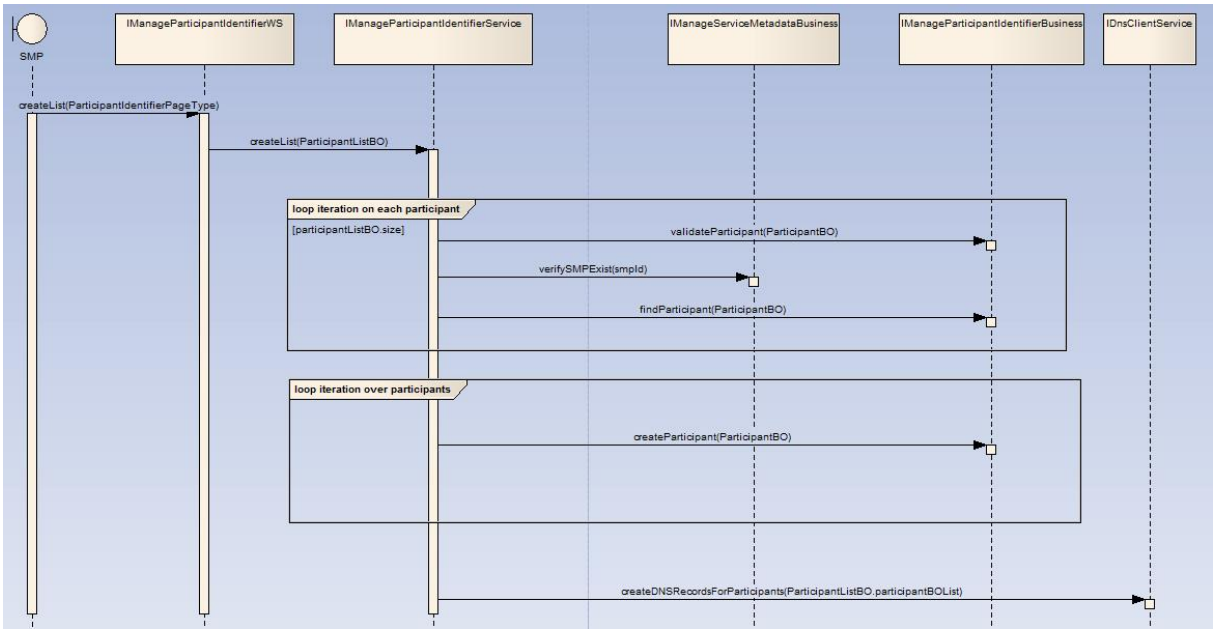

Figure 7 - Sequence Diagram - ManageParticipantIdentifier CreateList

3.3.3.2.4. Operation Delete()

3.3.3.2.4.1. Pre-requisites

- The user has a valid certificate
- The SMP already exists
- The role of the user is ROLE_SMP
- The participant already exists

3.3.3.2.4.2. Description

Deletes the information that the SML Service holds for a specific Participant Identifier.

- Input DeleteParticipantIdentifier: ServiceMetadataPublisherServiceForParticipantType - contains the Participant Identifier for a given participant and the identifier of the SMP that publishes its metadata
- Fault: notFoundFault - returned if the participant identifier or the identifier of the SMP could not be found
- Fault: unauthorizedFault - returned if the caller is not authorized to invoke the Delete operation
- Fault: badRequestFault - returned if the supplied DeleteParticipantIdentifier does not contain consistent data
- Fault: internalErrorFault - returned if the SML service is unable to process the request for any reason

3.3.3.2.4.3. Technical Design

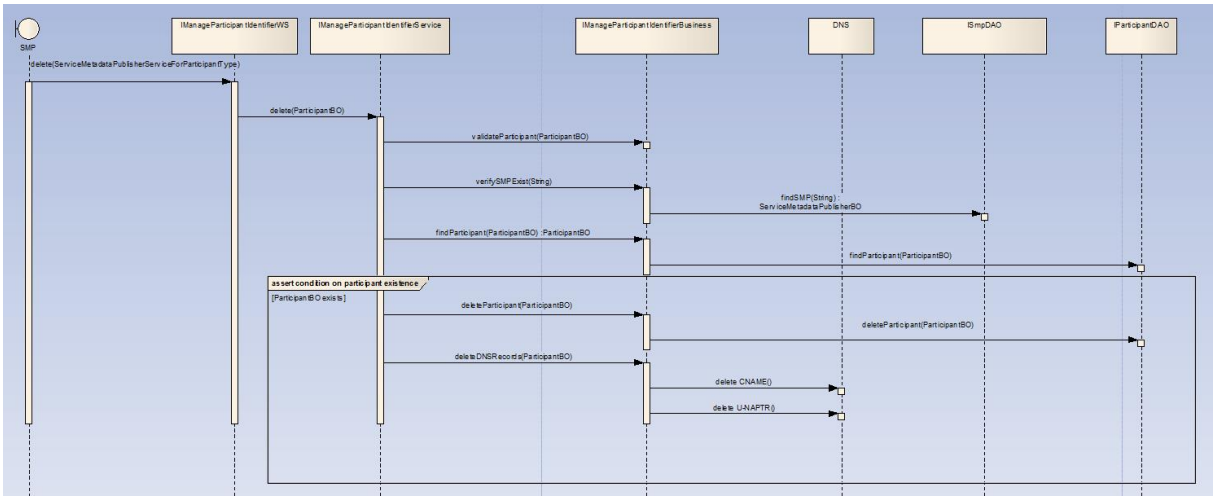

Figure 8 - Sequence Diagram - ManageParticipantIdentifier Delete

3.3.3.2.5. Operation DeleteList()

3.3.3.2.5.1. Pre-requisites

- The user has a valid certificate
- The SMP already exists
- The participants already exists
- The role of the user is ROLE_SMP
- The number of participants in the list is less than 100

3.3.3.2.5.2. Description

Deletes the information that the SML Service holds for a list of Participant Identifiers.

- Input DeleteList: ParticipantIdentifier - contains the list of Participant Identifiers for the participants which are removed from the Service Metadata Locator Service. The NextPageIdentifier element is absent.
- Fault: notFoundFault - returned if one or more participant identifiers or the identifier of the SMP could not be found
- Fault: unauthorizedFault - returned if the caller is not authorized to invoke the DeleteList operation
- Fault: badRequestFault - returned if:
- The supplied DeleteList does not contain consistent data
- The number of participants in the list is greater than 100
- Fault: internalErrorFault - returned if the SML service is unable to process the request for any reason

3.3.3.2.5.3. Technical Design

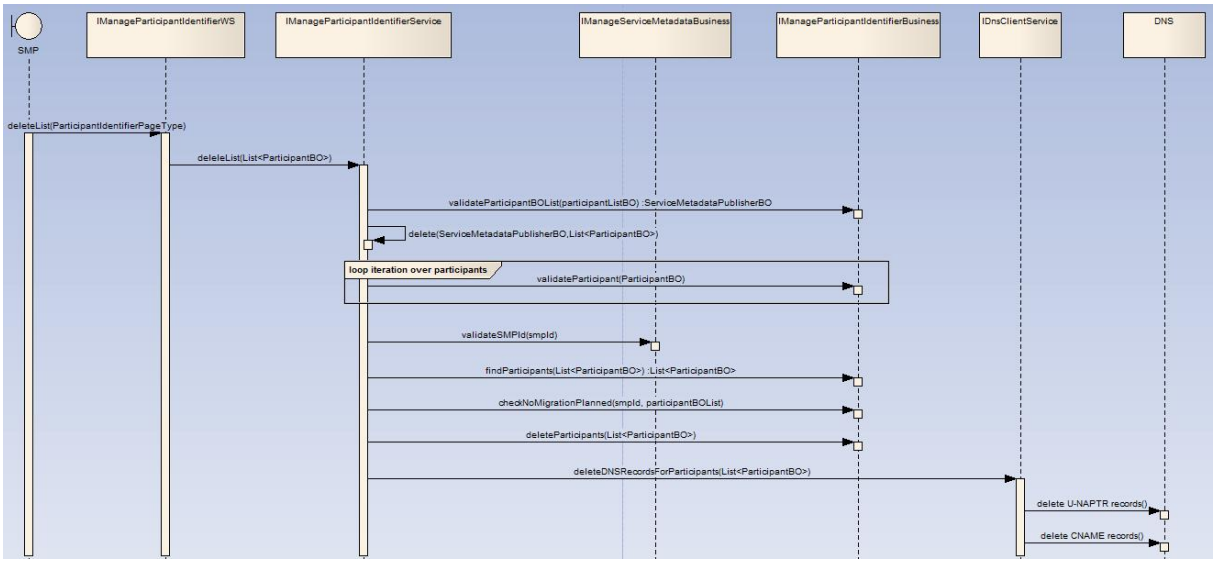

Figure 9 - Sequence Diagram - ManageParticipantIdentifier DeleteList

3.3.3.2.6. Operation PrepareToMigrate()

3.3.3.2.6.1. Pre-requisites

- The user has a valid certificate
- The SMP already exists
- The role of the user is ROLE_SMP
- The participant already exists

3.3.3.2.6.2. Description

Prepares a Participant Identifier for migration to a new Service Metadata Publisher. This operation is called by the Service Metadata Publisher which currently publishes the metadata for the Participant Identifier. The Service Metadata Publisher supplies a Migration Code which is used to control the migration process. The Migration Code must be passed (out of band) to the Service Metadata Publisher which is taking over the publishing of the metadata for the Participant Identifier and which MUST be used on the invocation of the Migrate() operation. This operation can only be invoked by the Service Metadata Publisher which currently publishes the metadata for the specified Participant Identifier.

- Input PrepareMigrationRecord: MigrationRecordType - contains the Migration Key and the Participant Identifier which is about to be migrated from one Service Metadata Publisher to another.
- Fault: notFoundFault - returned if the participant identifier or the identifier of the SMP could not be found
- Fault: unauthorizedFault - returned if the caller is not authorized to invoke the PrepareToMigrate operation
- Fault: badRequestFault - returned if the supplied PrepateMigrationRecord does not contain consistent data
- Fault: internalErrorFault - returned if the SML service is unable to process the request for any reason

3.3.3.2.6.3. Technical Design

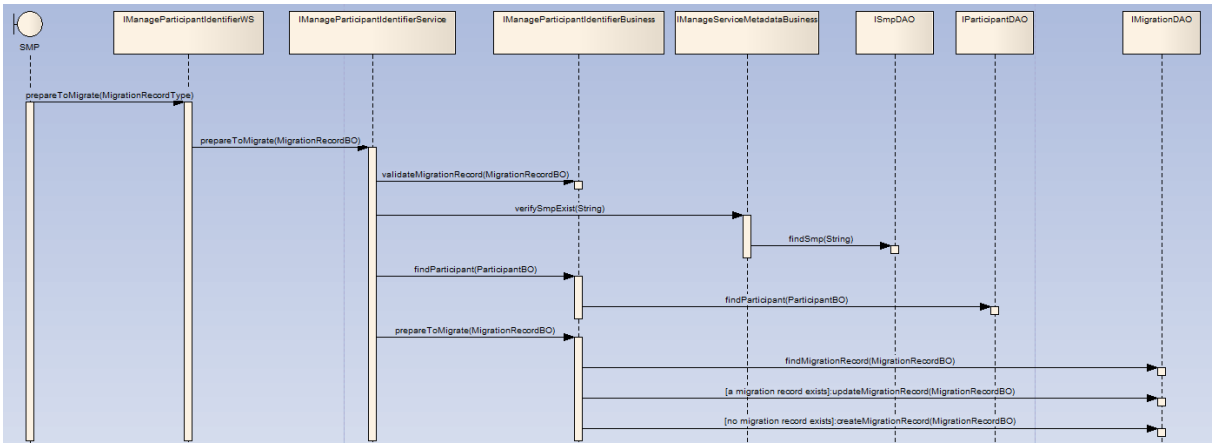

Figure 10 - Sequence Diagram - ManageParticipantIdentifier PrepareToMigrate

3.3.3.2.7. Operation Migrate()

3.3.3.2.7.1. Pre-requisites

- The user has a valid certificate
- The SMP already exists
- The participant already exists
- The role of the user is ROLE_SMP
- The prepareToMigrate service has been called for this participant

3.3.3.2.7.2. Description

Migrates a Participant Identifier already held by the Service Metadata Locator Service to target a new Service Metadata Publisher. This operation is called by the Service Metadata Publisher which is taking over the publishing for the Participant Identifier. The operation requires the new Service Metadata Publisher to provide a migration code which was originally obtained from the old Service Metadata Publisher. The PrepareToMigrate operation MUST have been previously invoked for the supplied Participant Identifier, using the same MigrationCode, otherwise the Migrate() operation fails. Following the successful invocation of this operation, the lookup of the metadata for the service endpoints relating to a particular Participant Identifier will resolve (via DNS) to the new Service Metadata Publisher.

- Input CompleteMigrationRecord: MigrationRecordType - contains the Migration Key and the Participant Identifier which is to be migrated from one Service Metadata Publisher to another.
- Fault: notFoundFault - returned if the migration key or the identifier of the SMP could not be found
- Fault: unauthorizedFault - returned if the caller is not authorized to invoke the Migrate operation
- Fault: badRequestFault - returned if the supplied CompleteMigrationRecord does not contain consistent data
- Fault: internalErrorFault - returned if the SML service is unable to process the request for any reason

3.3.3.2.7.3. Technical Design

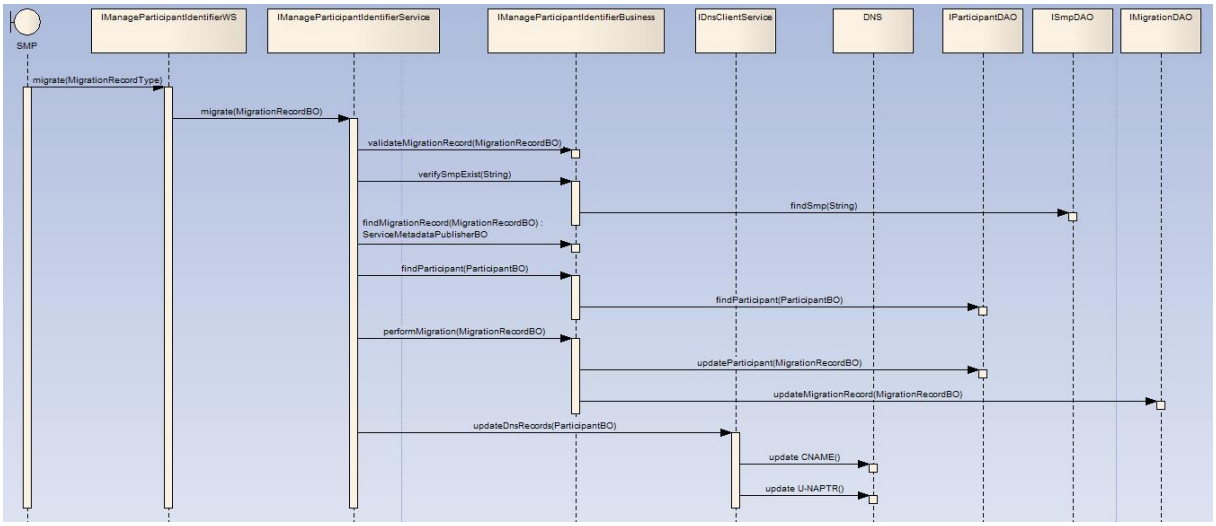

Figure 11 - Sequence Diagram - ManageParticipantIdentifier Migrate

3.3.3.2.8. Operation List()

3.3.3.2.8.1. Pre-requisites

- The user has a valid certificate
- The role of the user is ROLE_SMP
- The SMP already exists

3.3.3.2.8.2. Description

List() is used to retrieve a list of all participant identifiers associated with a single Service Metadata Publisher, for synchronization purposes. Since this list may be large, it is returned as pages of data, with each page being linked from the previous page.

- Input Page: PageRequest - contains a PageRequest containing the ServiceMetadataPublisherID of the SMP and (if required) an identifier representing the next page of data to retrieve. If the NextPageIdentifier is absent, the first page is returned.
- Output: ParticipantIdentifierPage - a page of Participant Identifier entries associated with the Service Metadata Publisher, also containing a <Page/> element containing the identifier that represents the next page, if any.
- Fault: notFoundFault - returned if the next page or the identifier of the SMP could not be found
- Fault: unauthorizedFault - returned if the caller is not authorized to invoke the List operation
- Fault: badRequestFault - returned if the supplied NextPage does not contain consistent data
- Fault: internalErrorFault - returned if the SML service is unable to process the request for any reason

Note that the underlying data may be updated between one invocation of List() and a subsequent invocation of List(), so that a set of retrieved pages of participant identifiers may not represent a consistent set of data.

3.3.3.2.8.3. Technical Design

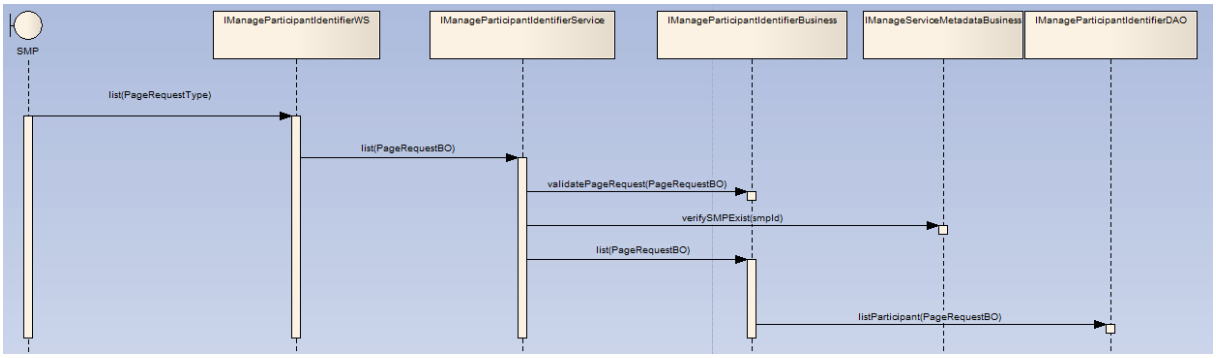

Figure 12 - Sequence Diagram - ManageParticipantIdentifier List

3.3.3.3. BDMSLService interface

This interface describes non-core services that are not defined in the SML or BDX specifications.

3.3.3.3.1. WSDL file

- BDMSLService-1.0.wsdl

3.3.3.3.2. Operation PrepareChangeCertificate()

3.3.3.3.2.1. Pre-requisites

- The current certificate of the user is valid
- The role of the user is ROLE_SMP
- The user has the new certificate for the SMP(s)

3.3.3.3.2.2. Description

This operation allows an SMP to prepare a change of its certificate. It is typically called when an SMP has a certificate that is about to expire and already has the new one. This operation MUST be called while the certificate that is already registered in the BDMSL is still valid. If the migrationDate is not empty, then the new certificate MUST be valid at the date provided in the migrationDate element. If the migrationDate element is empty, then the "Valid From" date is extracted from the certificate and is used as the migrationDate. In this case, the "Not Before" date of the certificate must be in the future.

- Fault: unauthorizedFault - returned if the caller is not authorized to invoke the PrepareChangeCertificate operation
- Fault: badRequestFault - returned if
  - The supplied request does not contain consistent data
  - The new certificate is not valid at the date provided in the migrationDate element
  - The migrationDate is not in the future.
  - The migrationDate is not provided and the "Not Before" date of the new certificate is not in the future
  - The migrationDate is not provided and the "Valid From" is in the past
- Fault: internalErrorFault - returned if the BDMSL service is unable to process the request for any reason

3.3.3.3.2.3. Technical design

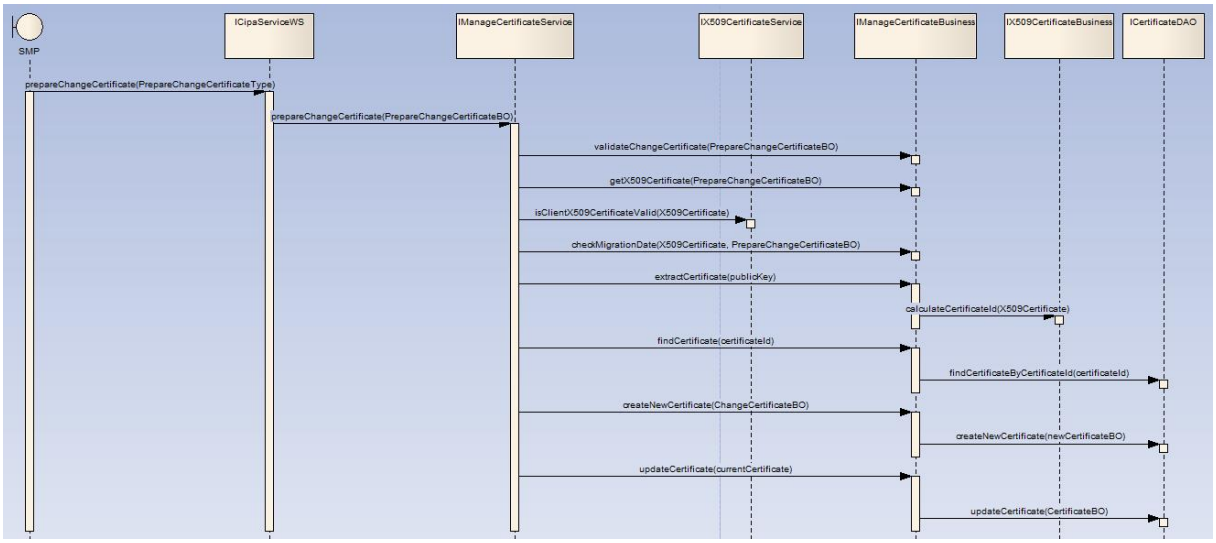

Figure 13 - Sequence Diagram - BDMSLService PrepareChangeCertificate()

3.3.3.3.2.4. Notes

A nightly job performs an analysis to actually perform the change of certificates. The algorithm is as following:

```
List<Certificate> certificates = findCertificateWithPessimisticLock()
for each certificate in certificates do
  if [certificate.new_cert_migration_date <= today] then
    for each allowed_wildcard in bdmsl.allowed_wildcard do
      allowed_wildcard.fk_certificate_id = certificate.new_cert_id
    end for
    for each smp in bdmsl.smp do
      smp.fk_certificate_id = certificate.new_cert_id
    end for
    delete certificate
  else if [certificate.new_cert_migration_date < today] then
    warn "The migration of the certificate couldn't be perform in time"
  end if
end for
```

The scheduling of the job can be configured by setting the value of the property `certificateChangeCronExpression`.

In order to avoid the job to be performed multiple times on a clustered environment, it is necessary to use a pessimistic lock when finding the certificates. The job must run in a single transaction and the lock is released at the end of the transaction.

3.3.3.3.3. Operation IsAlive()

3.3.3.3.3.1. Pre-requisites

- The certificate is valid
- The user has the role ROLE_SMP or ROLE_ADMIN

3.3.3.3.3.2. Description

This service has only a monitoring purpose. It can be called to check if the application is up and running.

This service checks if the database and the DNS are accessible by trying to read from the database and to write to and read from DNS.

- Input : none
- Output : none. HTTP 200 OK expected
- Fault: internalErrorFault - returned if the BDMSL service is unable to process the request for any reason

3.3.3.3.4. Operation ClearCache()

3.3.3.3.4.1. Pre-requisites

- The certificate is a valid certificate
- The user has the role ROLE_SMP or ROLE_ADMIN

3.3.3.3.4.2. Description

The application manages in-memory caches in order to enhance performances. This service can be called to clear all the caches managed by the application. The in-memory caches are used for:

- The list of trusted aliases and their corresponding domains, because these data are not supposed to be changed frequently
- The content of the Certificate Revocation List, in order to avoid the cost of downloading each time the CRLM for each certificate

- Input : none
- Output : none. HTTP 200 OK expected
- Fault: internalErrorFault - returned if the BDMSL service is unable to process the request for any reason

3.3.3.3.4.3. Technical design

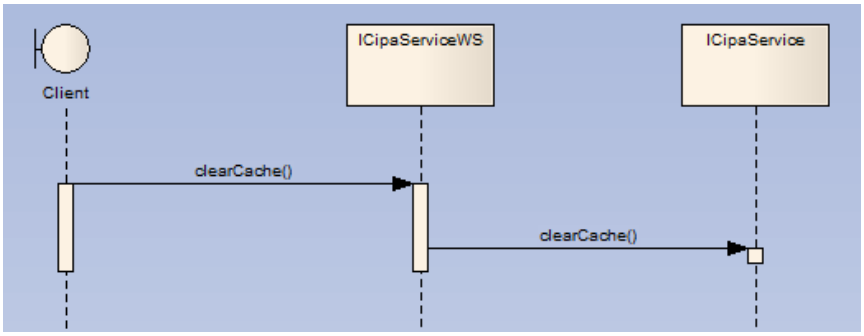

Figure 14 - Sequence Diagram – BDMSLService ClearCache()

3.3.3.3.5. Operation ListParticipants()

3.3.3.3.5.1. Pre-requisites

- The certificate is valid
- The user has the role ROLE_PYP

3.3.3.3.5.2. Description

Lists all the participants managed by the BDMSL. This service is only meant to be called by the PEPPOL Yellow Pages application.

- Input : none
- Output : ListParticipantsType : the complete list of the participants managed by the BDMSL
- Fault: UnauthorizedFault : Returned if the certificate provided is not a PYP certificate
- Fault: InternalFaultError : Returned if the BDMSL service is unable to process the request for any reason

3.3.3.3.5.3. Technical design

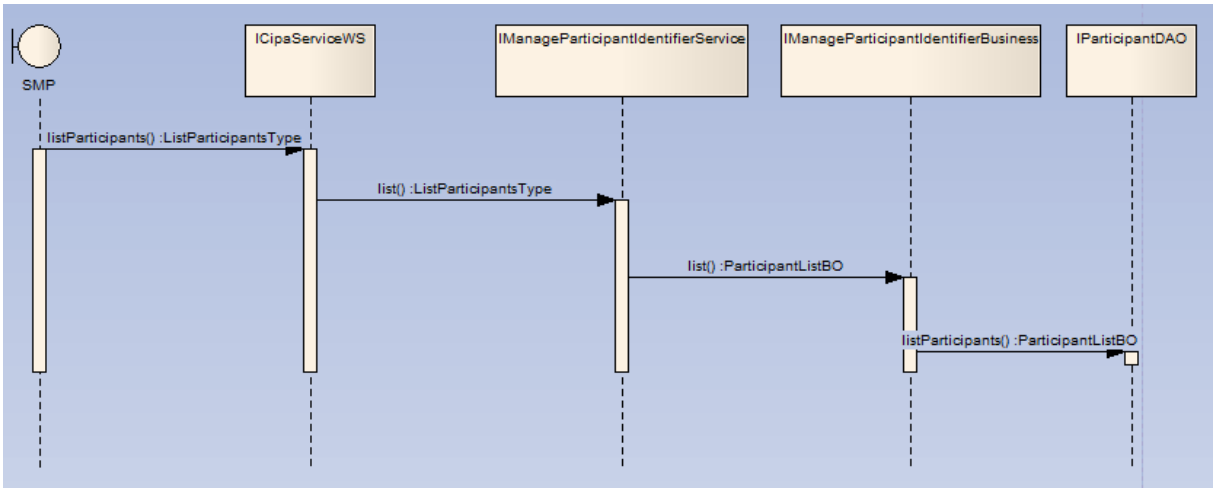

Figure 15 - Sequence Diagram – BDMSLService ListParticipants()

3.3.3.3.6. Operation CreateParticipantIdentifier()

3.3.3.3.6.1. Pre-requisites

- The certificate is valid
- The SMP already exists
- The participant doesn't already exists

3.3.3.3.6.2. Description

This service has the same behaviour as the `Create()` operation in the `ManageParticipantIdentifier` interface but it has one additional and optional input: the `serviceName` element. In the `Create`() operation, the service name is "`Meta:SMP`" by default. In the `CreateParticipantIdentifier()` operation, this service name can be customized.

- serviceName: the name of the service for the NAPTR record

3.3.3.3.6.3. Technical Design

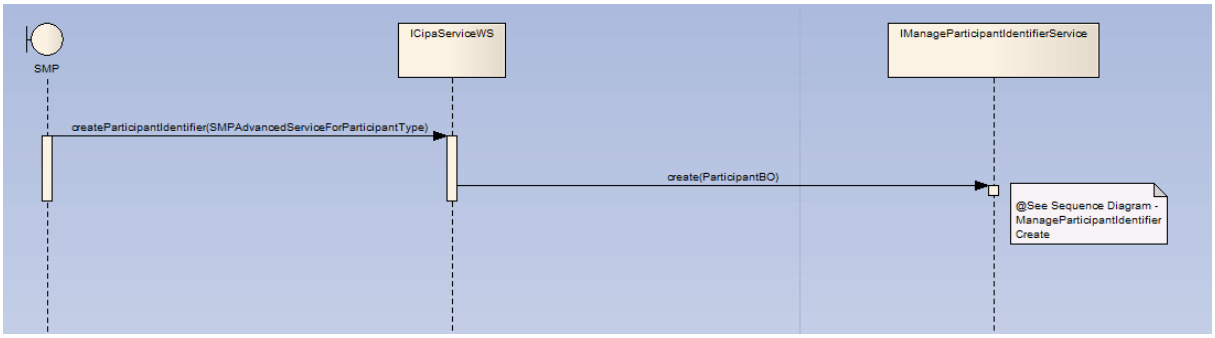

Figure 16 - Sequence Diagram - BDMSLService CreateParticipantIdentifier()

Note: the flow for the `create` method of `ManageParticipantIdentifierServiceImpl` can be found here: 3.3.3.2.2.3 Technical Design

3.3.3.3.7. Operation ChangeCertificate()

3.3.3.3.7.1. Pre-requisites

- The user credentials are valid
- The user has the role ROLE_ADMIN
- The user has the new certificate for the SMP

3.3.3.3.7.2. Description

This operation allows the admin team to change the SMP's certificate. It is called by the admin team in case the SMP's certificate has expired and the new one needs to be applied. The new certificate MUST be valid at the date time the request is sent.

- Input : SMP id, Certificate public key
- Output : none. HTTP 200 OK expected
- Fault: unauthorizedFault - returned if:
  - The caller is not authorized to invoke the ChangeCertificate operation (The user doesn't have the ROLE_ADMIN role)
  - The public key already exists
- Fault: badRequestFault - returned if
  - The supplied request does not contain consistent data
  - Invalid public key
  - The new certificate is not valid at the moment the request is sent
  - The SMP id is unknown
- Fault: internalErrorFault - returned if the BDMSL service is unable to process the request for some reason

3.3.3.3.7.3. Technical design

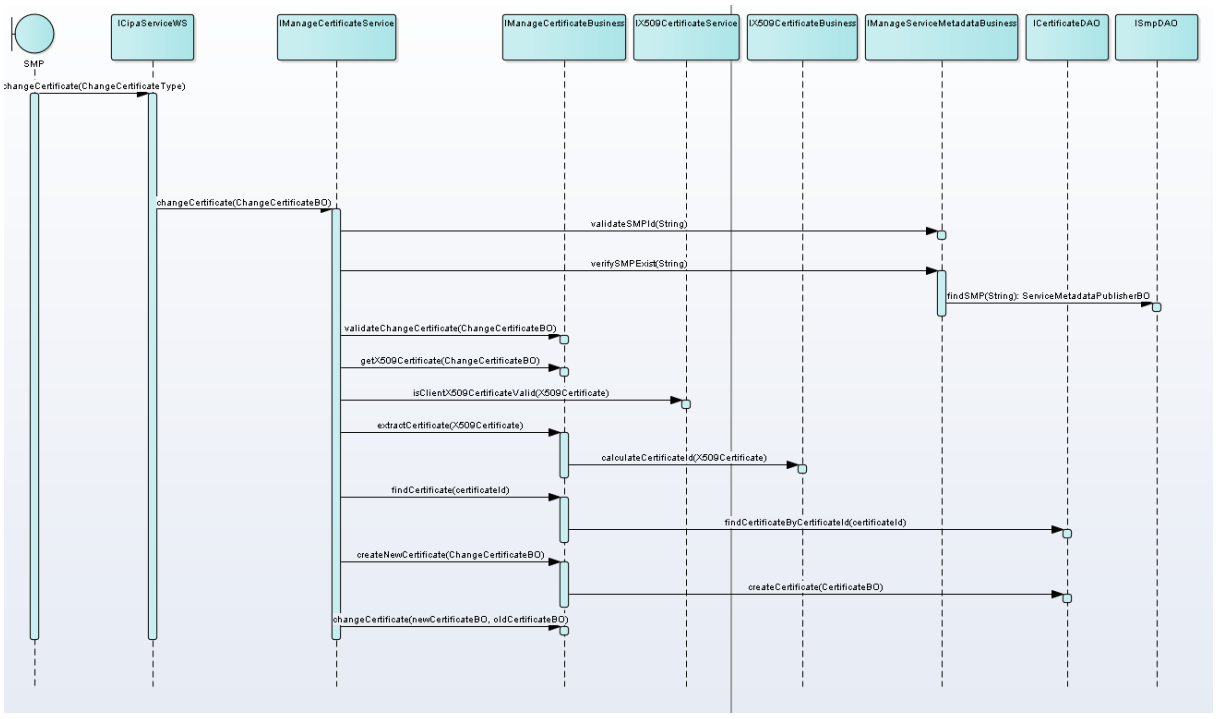

Figure 16 - Sequence Diagram - BDMSLService ChangeCertificate()

3.4. Business layer

The business layer manipulates only business objects and defines the business rules.

Business objects are POJO and implement the Singleton pattern. They are defined in the common package described in section 3.6.3 Business Objects (BO).

3.4.1. Naming convention

- Package: eu.europa.ec.bdmsl.business
- Interface: eu.europa.ec.bdmsl.business.I<InterfaceName>Business
- Implementation package: eu.europa.ec.bdmsl.business.impl
- Implementation: eu.europa.ec.bdmsl.business.impl.<InterfaceName>BusinessImpl

3.4.2. Dependencies

In this layer, only the following calls are allowed:

- Calls to technical components
- Calls to the persistence layer
- Calls to the common package

3.4.3. Frameworks

This layer handles most of the business logic processing. Therefore, there is no technical aspect. The only framework used is Spring for the dependency injection.

3.4.4. Development of the business layer

3.4.4.1. Interface

```
public interface IManageServiceMetadataBusiness {

  /**
   * Retrieves the Service Metadata Publisher record for the service metadata.
   * @param serviceMetadataPublisherID the unique ID
   *          of the Service Metadata Publisher for which the record is required
   * @return ServiceMetadataPublisherBO the service metadata publisher record.
   * @throws TechnicalException Technical exception.
   * @throws BusinessException Business exception.
   */
  ServiceMetadataPublisherBO read(String serviceMetadataPublisherID);
}
```

3.4.4.2. Implementation classes

The implementation classes extend the parent-class «`AbstractBusinessImpl`» and implement their dedicated interface (here «`IManageServiceMetadataBusiness`»). The `AbstractBusinessImpl` class only contains the logging service but may be completed with new services if new common requirements are identified for `*BusinessImpl` classes in future versions.

```
public class ManageServiceMetadataBusinessImpl extends AbstractBusinessImpl
implements IManageServiceMetadataBusiness {
```

```
  private IManageServiceMetadataDAO manageServiceMetadataDAO;

  /*
   * (non-Javadoc)
   *
   * @see eu.europa.ec.bdmsl.service.IManageServiceMetadataBusiness#read (String)
   */
  @Override
  public ServiceMetadataPublisherBO read(String serviceMetadataPublisherID)
    throws TechnicalException, BusinessException {
    ServiceMetadataPublisherBO smpBO =
manageServiceMetadataServiceBusiness.read(serviceMetadataPublisherID);
    return smpBO;
  }
  ...
}
```

3.5. Data Access layer

This layer access to the data persisted in the database. The objects of this layer are POJO that implement the Singleton and DAO pattern.

3.5.1. Naming convention

- Package: eu.europa.ec.bdmsl.dao
- Interface: eu.europa.ec.bdmsl.dao.I<InterfaceName>DAO
- Implementation package: eu.europa.ec.bdmsl.dao.impl
- Implementation: eu.europa.ec.bdmsl.dao.impl.<InterfaceName>DAOImpl
- Package Entity Object: eu.europa.ec.bdmsl.dao.entity
- Entity object: eu.europa.ec.bdmsl.dao.entity.<ObjectName>Entity

3.5.2. Dependencies

In this layer, only the following calls are allowed:

- Calls to technical components
- Calls to the common package

3.5.3. Frameworks

This layer is the only one to use the JPA framework because it is the only one that actually accesses to the database.

The configuration is managed by Spring.

3.5.4. Development of the data access layer

3.5.4.1. Interface

```
public interface ISmpDAO {

  /**
   * Retrieves the Service Metadata Publisher record for the service metadata.
   * @param serviceMetadataPublisherID the unique ID
```

```
    *           of the Service Metadata Publisher for which the record is required
    * @return ServiceMetadataPublisherBO the service metadata publisher
    */
   ServiceMetadataPublisherBO findSMP(String serviceMetadataPublisherID) throws
TechnicalException;

}
```

3.5.4.2. Implementation classes

The implementation classes extend the parent-class «`AbstractDAOImpl`» and implement their dedicated interface (here «`IManageServiceMetadataDAO`»).

```
public class ManageServiceMetadataDAOImpl extends AbstractDAOImpl implements
ISmpDAO {

   /**
    * @see eu.europa.ec.bdmsl.dao.ISmpDAO#findSMP(String)
    */
   @Override
   public ServiceMetadataPublisherBO findSMP(String serviceMetadataPublisherID)
     throws TechnicalException {
            ServiceMetadataPublisherBO resultBO = null;
         SmpEntity resultSmpEntity = getEntityManager().find(SmpEntity.class, id);
         if (resultSmpEntity != null) {
            resultBO = mapperFactory.getMapperFacade().map(resultSmpEntity,
ServiceMetadataPublisherBO.class);
         } else {
            loggingService.debug("No SMP found for id " + id);
         }

         return resultBO;
    }
...
}
```

3.5.4.3. Mapping BO / entity

The data access layer internally uses JPA entities to perform the Object/Relational mapping with the database. However, the methods exposed in the interfaces only expose Business Objects because the Business objects are the only ones that can be used between the layers. For more information on the mapping BO/Entities, see the chapter ***8 Object mapping.***

3.6. Common package

This package is particular because it can be called without restriction by all the layers of the application: it is transversal.

This common package provides:

- Business and technical Exceptions.
- Business objects (BOs) to be used in every layer
- Constants, error codes, utility classes
- Enums

3.6.1. Naming convention

- Package: eu.europa.ec.bdmsl.common

- Package Business Object: eu.europa.ec.bdmsl.common.bo
- Business Object: eu.europa.ec.bdmsl.common.<ObjectName>BO

3.6.2. Dependencies

In this layer, only the following calls are allowed:

- Calls to technical components

3.6.3. Business Objects (BO)

Business objects (BO) are developed in the common package because they are transversal to all layers and are used in the service, business and persistence layer. They are POJO with no dependency to any framework or database. They can walk through the layers. We use BO because they are linked to the domain, and hide the implementation choices made for façade and for the persistence. Thus, they are not directly linked to any database model, or any web service interface.

Each BO extends the abstract class `AbstractBusinessObject` provided by the common library. This class implements `java.io.Serializable` and overrides `equals`, `hashCode` and `toString` as abstract methods.

Each BO must define a `serialVersionUID` and implement the 3 previous methods.

4. SOFTWARE ARCHITECTURE

There are 4 different maven projects:

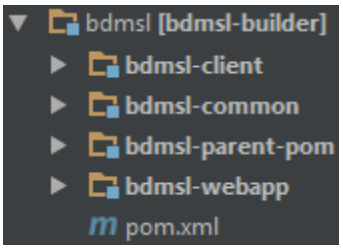

Figure 17 - Project structure

In this chapter, we describe the content and the role of each project.

4.1. Library "bdmsl-common"

Project name : bdmsl-common

Packages:

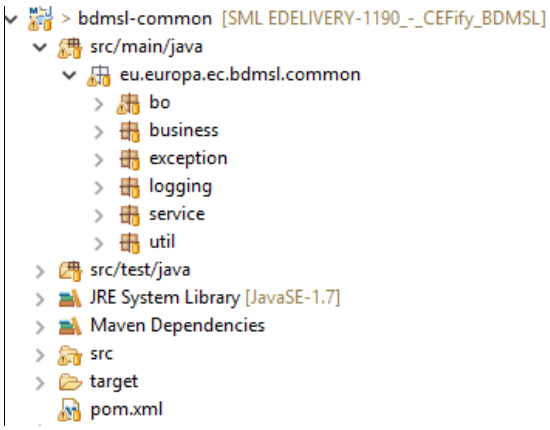

Figure 18 - Packages of the bdmsl-common project

This library is used by all the modules of the eDelivery BDMSL solution. It provides services like:

- Cryptography
- Constants
- Configuration manager
- Utils (dates, encoding, etc.)
- Logging
- Abstract/parent classes common to all eDelivery BDMSL modules

4.2. bdmsl-webapp

Project name : bdmsl-webapp

This is a Maven project that contains all the services, business logic and persistence code of the application. It produces a `war` file that can be deployed in the supported application servers and servlet containers.

There are multiple profiles of compilation:

- `weblogic-oracle` : produces a war compatible with a Weblogic application server and an Oracle database
- `tomcat-mysql` : produces a war compatible with a tomcat servlet container and a Mysql database
- `jboss-mysql` : produces a war compatible with a JBoss application server and a Mysql database. Currently, this profile still needs to be configured.

To produce a war compatible with the profile `weblogic-oracle`, run the following command at the root of the `bdmsl-webapp` folder:

```
mvn clean install -Pweblogic-oracle
```

4.3. Web service client

Project name: bdmsl-client

It's a SOAP web service client (stub) that is generated from the WSDLs of the main project. The client is a Maven project and the output is packaged as a `jar` file.

The WSDL files contain all the methods that are exposed, the objects and the exceptions.

The projects that call the web services of the eDelivery BDMSL application can use this web service client.

4.4. Parent pom

Project name: bdmsl-parent-pom

It's the parent pom of all the Maven module. It contains the version for the dependencies, default configuration of plugins, etc.

4.5. Maven configuration

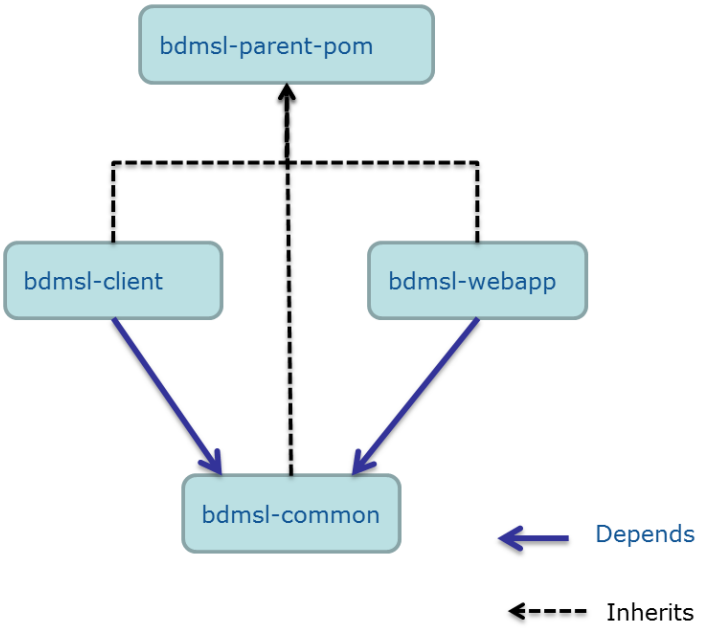

Figure 19 - Dependency tree of the maven projects

5. LOGGING

5.1. Implementation

The logs use the Log4j framework. The `bdmsl-common` library provides the logging manager `ILoggingService` and its main implementation class `LoggingServiceImpl`. This logging manager must be used for all the logs within the eDelivery BDMSL application.

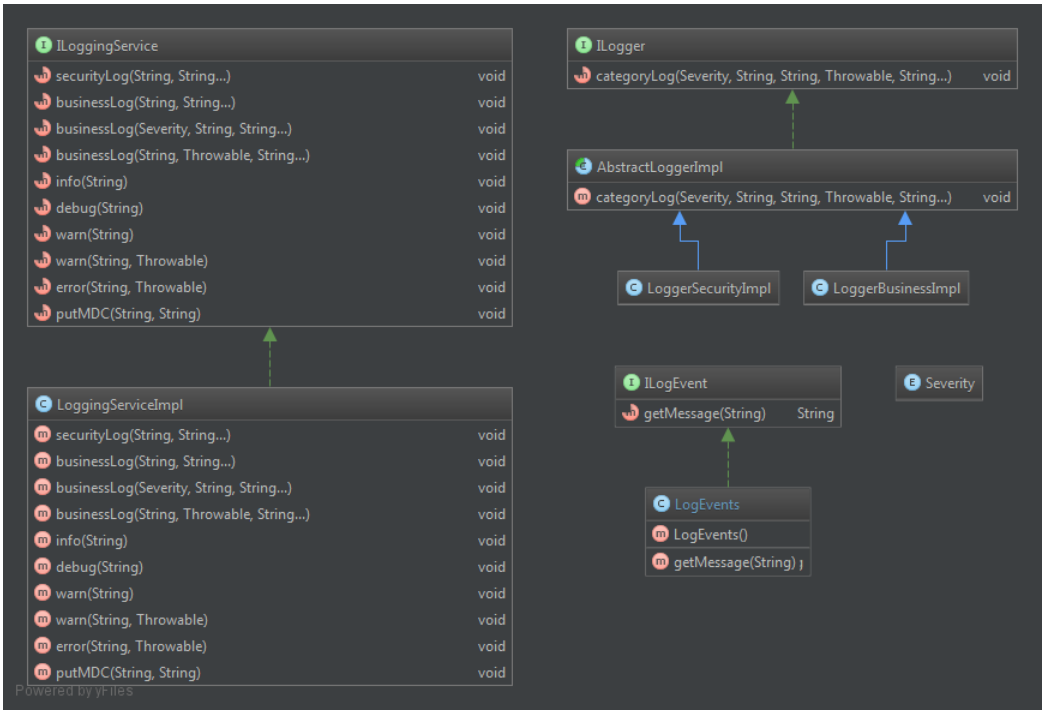

Figure 20 - Logging class diagram

There are 3 types of logs: security logs, business logs and miscellaneous logs. Each category of log has its own appender defined in the `log4j.xml` file. By default, each category will log in a separate file:

- `bdmsl-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.
- `bdmsl-business.log`: This log file contains all the business related information. For example, when a participat is created, when a SMP is deleted, etc.
- `bdmsl.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 implementation of the `ILogEvent` interface. In the eDelivery BDMSL application, all the security and business messages are defined in the `LogEvents` class.

The pattern of the logs is defined in the `log4j.xml` file. The default pattern is:

```
%d{ISO8601}{Europe/Brussels} [%X{user}] [%X{requestId}] %-5p %c{1}:%L - %m%n
```

- `user`: The client authenticated by its certificate.
- `requestId`: the UUID of the request (provided by the application server)

The values for the `user` and `requestId` properties can be set by calling the method `ILoggingService.putMDC(String key, String value)`.

5.2. Log event codes

Category	Log event code	Description
SECURITY	SEC-001	The host %s attempted to access %s without any certificate
SECURITY	SEC-002	The host %s has been granted access to %s with roles %s
SECURITY	SEC-003	The host %s has been refused access to %s
SECURITY	SEC-004	The certificate is revoked : %s
SECURITY	SEC-005	The root certificate of the client certificate is unknown in the database. It means that the certificate is accepted at transport level (SSL) but refused at application level. %s
SECURITY	SEC-006	Certificate is not valid at the current date %s. Certificate valid from %s to %s
SECURITY	SEC-007	Certificate is not yet valid at the current date %s. Certificate valid from %s to %s
BUSINESS	BUS-001	Technical error while authentication process
BUSINESS	BUS-002	Error while configuring the application.
BUSINESS	BUS-003	The SMP was successfully created: %s.
BUSINESS	BUS-004	The SMP couldn't be created: %s.
BUSINESS	BUS-005	The following SMP was read: %s.
BUSINESS	BUS-006	The SMP couldn't be read: %s.
BUSINESS	BUS-007	The SMP was successfully deleted: %s.
BUSINESS	BUS-008	The SMP couldn't be deleted: %s.
BUSINESS	BUS-009	The SMP was successfully updated: %s.
BUSINESS	BUS-010	The SMP couldn't be updated: %s.
BUSINESS	BUS-011	The participant was successfully created: %s.
BUSINESS	BUS-012	The participant couldn't be created: %s.
BUSINESS	BUS-013	The list of participant couldn't be created: %s.
BUSINESS	BUS-014	The list of participants couldn't be created: %s.
BUSINESS	BUS-015	The participant was successfully deleted: %s.
BUSINESS	BUS-016	The participant couldn't be deleted: %s.
BUSINESS	BUS-017	The list of participant couldn't be deleted: %s.
BUSINESS	BUS-018	The list of participants couldn't be deleted: %s.
BUSINESS	BUS-019	The participants of SMP %s have been successfully listed.
BUSINESS	BUS-020	The participants of SMP %s couldn't be listed.
BUSINESS	BUS-021	The prepare to migrate service was successfully called for participant: %s.
BUSINESS	BUS-022	The prepare to migrate service failed for participant: %s.
BUSINESS	BUS-023	The call to migrate service was successfully called for participant: %s.
BUSINESS	BUS-024	The call to migrate service failed for participant: %s.
BUSINESS	BUS-025	The call to the list service succeeded
BUSINESS	BUS-026	The call to the list service failed
BUSINESS	BUS-027	The new certificate was successfully planned for change for current certificate: %s
BUSINESS	BUS-028	The certificate change failed for current certificate: %s
BUSINESS	BUS-029	The following CNAME record has been added to the DNS for the participant %s : %s

BUSINESS	BUS-030	The following NAPTR record has been added to the DNS for the participant %s : %s
BUSINESS	BUS-031	The following CNAME record has been added to the DNS for the SMP %s : %s
BUSINESS	BUS-032	The following A record has been added to the DNS for the SMP %s : %s
BUSINESS	BUS-033	The CertificateChangeJob ran successfully. %s certificates have been migrated
BUSINESS	BUS-034	The CertificateChangeJob failed.
BUSINESS	BUS-035	The ChangeCertificate service has been executed successfully
BUSINESS	BUS-036	The ChangeCertificate service has failed

Table 1 - Log event codes

6. CACHING

In order to enhance performance, in-memory caches are used in the application. They rely on the `ehcache` implementation. To put objects in a cache, we use annotations:

```
@Override
@Cacheable(value = "crlByUrl", key = "#crlDistributionPointURL")
public void verifyCertificateCRLs(String serial, String crlDistributionPointURL){
   [...]
}
```

The `@Cacheable` annotation triggers cache population. In the previous example, the name of the cache is `crlByUrl`. The `key` attribute is one of the parameters of the method: `crlDistributionPointURL`. The next time this method is called, if the cache is already populated with a value for the given key, then the method won't actually be called and the result will be returned from the cache.

Sometimes, it is useful to clear the caches. This can be done by calling the method `IBDMSLService.clearCache()`.

7. EXCEPTION HANDLING

7.1. Exception types

When exceptions are thrown in the business, persistence and service layers, they are transformed into technical or business exceptions to ensure to the client of the service that all the possible exceptions are declared in the service signature.

All the methods of the exposed interfaces in the persistence, business and service layer can only throw two kinds of exceptions:

- `TechnicalException` : Technical exceptions happen when a technical component of a business process acts in an unexpected way. Examples of technical exceptions are: IO exception, timeout, bad configuration, etc.
- `BusinessException` : Business Exceptions are exceptions that are designed and managed in the specification of a business process. In other words, Business Exceptions are exceptions which happen at the process or workflow level, and are not related to the technical components.

7.2. SOAP Faults

Because of the design of the WSDL in the SML specification, it is not possible to use an interceptor to transform the exceptions into SOAP fault. Thus, it is the `AbstractWSImpl` class which handles exceptions and convert any type of exception into appropriate SOAP faults. In the eDelivery BDMSL, there are 4 types of SOAP faults, all mapped to `TechnicalException`:

- NotFoundFault
- UnauthorizedFault
- BadRequestFault
- InternalErrorFault

A typical SOAP fault example would be:

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
   <soap:Body>
      <soap:Fault>
         <faultcode>soap:Server</faultcode>
         <faultstring>5378C6571DE2DD3FD026704338FF678B</faultstring>
         <detail>
            <NotFoundFault
xmlns:ns2="http://busdox.org/transport/identifiers/1.0/"
xmlns="http://busdox.org/serviceMetadata/locator/1.0/">
               <FaultMessage>[ERR-100] The SMP 'testSMLUpdate' doesn't
exist.</FaultMessage>
            </NotFoundFault>
         </detail>
      </soap:Fault>
   </soap:Body>
</soap:Envelope>
```

In the previous SOAP fault, the `faultstring` contains the request unique identifier provided by the application server. This request unique identifier is traced in the logs, in order to easily find the logs associated to an exception:

```
2015-07-24 10:32:08,562 [unsecure-http-client] [5378C6571DE2DD3FD026704338FF678B]
ERROR LoggingServiceImpl:83 - [ERR-100] The SMP 'testSMLUpdate' doesn't exist.
```

The error codes are all listed in the `IErrorCodes` interface (see table in the following paragraph).

7.3. Error codes

Error code	Description	Exception type
100	SMP not found error	TechnicalException
101	Unauthorized error	TechnicalException
102	Certificate authentication issue	TechnicalException
103	The root alias is not found in the list of trusted issuers in the database	TechnicalException
104	The certificate is revoked	TechnicalException
105	Generic technical error	TechnicalException
106	Bad request error	TechnicalException
107	DNS communication problem	TechnicalException
108	Problem with the SIG0 signature	TechnicalException
109	Bad configuration	TechnicalException
110	Participant not found error	TechnicalException
111	Migration data not found	TechnicalException
112	Duplicate participant error	TechnicalException
113	Error when deleting a SMP	TechnicalException
114	The deletion failed because a migration is planned for the given participant or SMP	TechnicalException
115	The certificate couldn't be found	TechnicalException

Table 2 - Error Codes

8. OBJECT MAPPING

There are 3 types of objects used in the application:

- JAXB objects : Generated objects from the WSDL
- Business objects (BO) : POJO used in the business logic in the service, business and persistence layers
- JPA entities : Persistence domain objects

2 types of mapping are required:

Figure 21 - Object mappings

The first type of mapping converts JAXB objects to BO and vice-versa. The implementation class is `SoapMappingInitializer`.

The second type of mapping converts JPA entities to BO and vice-versa. The implementation class is `EntityMappingInitializer`.

In order to avoid hand coding value object assemblers to copy data from one object type to another, we use a generic framework named Orika. Orika is a Java Bean mapping framework that recursively copies data from one object to another.

An example of mapping would be:

```
@Component
public class SoapMappingInitializer {

    @Autowired
    private MapperFactory mapperFactory;

    @PostConstruct
    public void init() {
            [...]
        mapperFactory.classMap(PageRequestType.class, PageRequestBO.class)
                .field("serviceMetadataPublisherID", "smpId")
                .byDefault()
                .register();
            [...]
    }
}
```

In the previous mapping, we map the field `serviceMetadataPublisherID` of the class `PageRequestType` to the field `smpId` of the class `PageRequestBO`. The other fields have the same name so they are automatically mapped thanks to the `byDefault()` method. This mapping is bidirectional.

To map an object, the singleton instance of the MapperFactory object can be used. For instance, in the Façade layer (ws) :

```
[...]

public class BDMSLServiceWSImpl extends AbstractWSImpl implements IBDMSLServiceWS
{
    [...]

    @Autowired
    private MapperFactory mapperFactory;
    [...]

    @Override
    @WebMethod([...])
    public void create(@WebParam ParticipantType participantType) {
     [...]
     // Convert the ParticipantType JAXB object into a ParticipantBO object
     ParticipantBO participantBO =
mapperFactory.getMapperFacade().map(participantType, ParticipantBO.class);
    [...]
    }
[...]
}
```

9. DATABASE MANAGEMENT

9.1. Auditing

In order to automatically audit the changes in the database, all the DAOs must extend the `AbstractDAOImpl` class and use its `persist()` and `merge()` methods. This way, the date of the changes of any business data is automatically logged.

For each table containing business data, these 2 following columns are present:

- `created_on`: date of creation of the row
- `last_updated_on`: date of the last update of the row

9.2. Versioning

`Liquibase` is used to manage the database versioning. `Liquibase` is an open source library for tracking, managing and applying database changes and can be used for any database.

The database scripts are written as `changeSet` in an XML file and the library then generates the appropriate SQL scripts for `Oracle`, `H2` or `MySQL` databases.

Each time the application starts, `Liquibase` checks the version of the database and executes the `changeSet` that have not been yet executed.

An example of `changeSet` would be:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<databaseChangeLog
        xmlns="http://www.liquibase.org/xml/ns/dbchangelog"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://www.liquibase.org/xml/ns/dbchangelog
         http://www.liquibase.org/xml/ns/dbchangelog/dbchangelog-3.3.xsd">

    <property name="autoIncrement" value="true" dbms="mysql,h2"/>
    <property name="autoIncrement" value="false" dbms="oracle"/>
    [...]
    <changeSet author="eDelivery" id="1">
        <createTable tableName="bdmsl_allowed_wildcard">
            <column name="scheme" remarks="The scheme to which the wildcard
applies" type="VARCHAR(255)">
                <constraints nullable="false"/>
            </column>
            <column name="fk_certificate_id" remarks="The foreign key to the
certificate" type="INT">
                <constraints nullable="false"/>
            </column>
            <column defaultValueComputed="${now}" name="created_on" remarks="Date
of creation"
                    type="datetime">
                <constraints nullable="false"/>
            </column>
            <column defaultValueComputed="${now}" name="last_updated_on"
remarks="Date of the last update"
                    type="datetime">
                <constraints nullable="false"/>
            </column>
```

```
        </createTable>
    </changeSet>
    [...]
    <changeSet author="eDelivery" id="8">
        <addPrimaryKey columnNames="fk_certificate_id, scheme"
constraintName="PRIMARY_AWS"
                       tableName="bdmsl_allowed_wildcard"/>
    </changeSet>
    [...]
</databaseChangeLog>
```

The `liquibase` scripts are located in the `src/main/resources/liquibase` folder. The main file is `db.changelog-master.xml`.

9.3. Data model

9.3.1. Overview

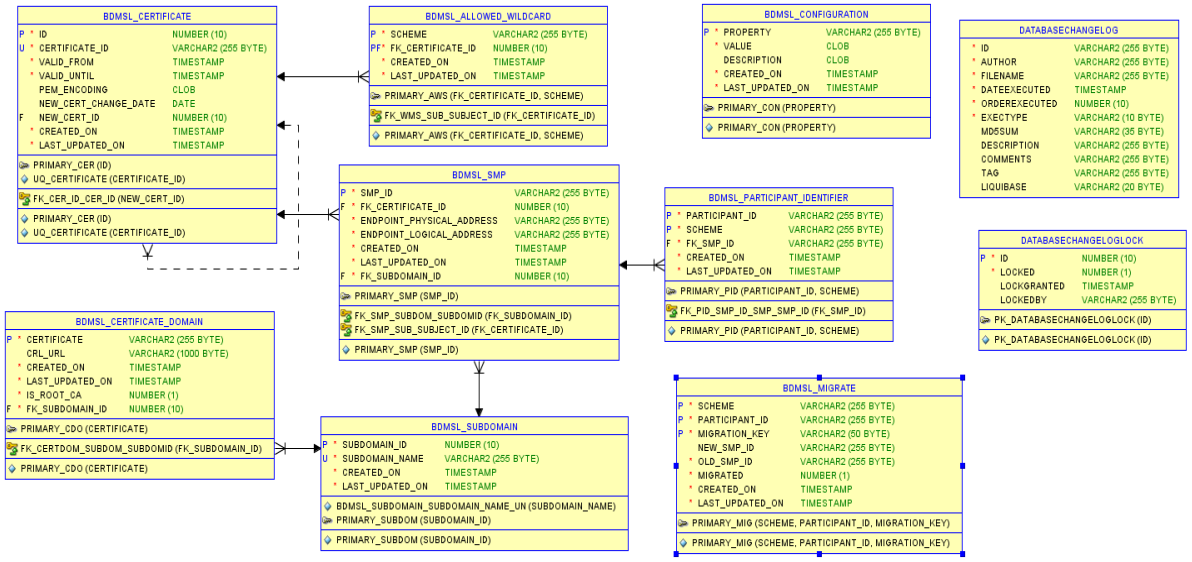

Figure 22 - Data model overview

9.3.2. Global description of the tables

Table	Description
bdmsl_allowed_wildcard	It is possible for a given Service Metadata Publisher to provide the metadata for all participant identifiers belonging to a particular participant identifier scheme. If this is the case, then it corresponds to the concept of a "wildcard" CNAME record in the DNS, along the lines: *.<schemeID>.<SML domain> CNAME <SMP domain><SMP domain> may either be the domain name associated with the SMP, or an alias for it. This implies that all participant identifiers for that schemeID will have addresses that resolve to the single address of that one SMP - and that as result only one SMP can handle the metadata for all participant identifiers of that scheme. Wildcard records are indicated through the use of "*" as the participant

	identifier in the operations of the ManageParticipantIdentifier interface. This table identifies the SMP with their certificates and map them to schemes for which they can create wildcard records.
bdmsl_certificate	List of SMPs identified with their certificates.
bdmsl_certificate_domain	Associates the root certificates to the DNS domains
bdmsl_configuration	Table containing all the configuration
bdmsl_migrate	Contains the participants migrated or to be migrated
bdmsl_participant_identifier	List of the participants
bdmsl_smp	List of the SMPs
Bdmsl_subdomain	List of Subdomains

Table 3 - Tables list

9.3.3. Detailed description of the tables

Table	Column	Description
bdmsl_allowed_wildcard	scheme	The scheme on which the wildcard applies
	fk_certificate_id	The foreign key to the certificate
	created_on	Date of creation
	last_updated_on	Date of the last update
bdmsl_certificate	certificate_id	The certificate_id is a key composed of the subject and the serial number of the certificate.
	valid_from	Start validity date of the certificate
	valid_until	Expiry date of the certificate
	pem_encoding	PEM encoding for the certificate
	new_cert_change_date	The date of the change for the new certificate
	new_cert_id	The new certificate id. Links to the certificate that will be valid after the current one is expired. At the migration date, it aims to replace the existing certificate
	created_on	Date of creation
	last_updated_on	Date of the last update
bdmsl_certificate_domain	certificate	Trusted root certificate. Must also be defined in the configured truststore
	fk_subdomain_id	The foreign key to the subdomain
	crl_url	URL to the certificate revocation list (CRL)
	created_on	Date of creation
	is_root_ca	If certificate is root CA or not
	last_updated_on	Date of the last update
bdmsl_configuration	property	Name of the property
	value	Value of the property
	description	Description of the property
	created_on	Date of creation
	last_updated_on	Date of the last update

bdmsl_migrate	scheme	The scheme of the participant identifier to be migrated
	participant_id	The participant identifier to be migrated
	migration_key	The migration key is a code that must be passed out-of-band to the SMP which is taking over the publishing of the metadata for the participant identifier. This code must contain: • 8 characters minimum • 24 characters maximum • 2 Special Characters @#$%()[]{}*^-!~|+= • 2 Upper Case letters minimum • 2 Lower Case letters minimum • 2 Numbers minimum • No white spaces
	new_smp_id	The id of the SMP after the migration
	old_smp_id	The id of the old SMP (before the migration)
	migrated	True if the migration is done
	created_on	Date of creation
	last_updated_on	Date of the last update
bdmsl_participant_identifier	participant_id	The participant identifier
	scheme	The scheme of the participant identifier
	fk_smp_id	The foreign key to the SMP identifier
	created_on	Date of creation
	last_updated_on	Date of the last update
bdmsl_smp	smp_id	The SMP identifier
	fk_certificate_id	The foreign key to the certificate
	endpoint_physical_address	The physical address of the endpoint. This physical address is used as the ALIAS on the CNAME DNS record.
	endpoint_logical_address	The logical address of the endpoint
	created_on	Date of creation
	fk_subdomain_id	The foreign key to the subdomain
	last_updated_on	Date of the last update
bdmsl_subdomain	subdomain_id	The subdomain identifier
	subdomain_name	The subdomain name
	created_on	Date of creation
	last_updated_on	Date of the last update

Table 4 - Tables fields

10. SCHEDULER

The Spring Framework provides abstractions for asynchronous execution and scheduling of tasks.

In the `applicationContext.xml` file, we can define the jobs to be scheduled:

```
<task:scheduler id="scheduler" pool-size="1"/>
<task:scheduled-tasks scheduler="scheduler">
    <task:scheduled ref="manageCertificateService" method="changeCertificates"
cron="0 0 2 ? * *"/>
</task:scheduled-tasks>
```

The previous example will execute every day at 2 am the method `changeCertificate` of the bean name `manageCertificateService`.

In case of the execution of the application on a clustered environment, it is necessary to make sure that multiple jobs won't perform the same task at the same time. The use of a pessimistic lock can be useful:

```
@Override
public List<CertificateBO> findCertificatesToChange(Calendar currentDate) throws
TechnicalException {
    // This method is used in the context of a job that can be run on a clustered
environment. To avoid concurrency issues, we do here a SELECT FOR UPDATE
    Query query = getEntityManager().createQuery("SELECT cert from
CertificateEntity cert where cert.newCertificateChangeDate <= :currentDate")
            .setParameter("currentDate",
currentDate).setLockMode(LockModeType.PESSIMISTIC_WRITE);

    [...]
}
```

10.1. Change Certificate

This job changes the certificates that have a migration date in the past or at the present day and deletes the older ones.

This task runs according to this parameter:

BDMSL_CONFIGURATION		
PROPERTY	**VALUE**	**DESCRIPTION**
certificateChangeCronExpression	0 0 2 ? * *	Cron expression for the changeCertificate job. Example: 0 0 2 ? * * (everyday at 2:00 am)

This parameter can be updated manually on the database or by means of a liquibase script.

10.2. Data Inconsistency Analyzer

This job looks for inconsistencies between the database and the DNS. It first accesses the DNS to retrieve all SMPs and Participants. It then compares DNS data against Database. All dismatch entries these changes are reported to the user by means of a report email.

As the previous job, this task will run according to the parameters below:

BDMSL_CONFIGURATION		
PROPERTY	**VALUE**	**DESCRIPTION**
dataInconsistencyAnalyzer.cronJobExpression	0 0 3 ? * *	Cron expression: 0 0 3 ? * * (every day at 3:00 am)
dataInconsistencyAnalyzer.recipientEmail	email@example.com	Email address to receive Data Inconsistency Checker results
dataInconsistencyAnalyzer.senderEmail	email@example.com	Sender email address for reporting Data Inconsistency Analyzer.

These parameters can be updated manually on the database or by means of a liquibase script.

11. VALIDATIONS

11.1. Participant ID validation per Domain

SML provides to each existent domain the possibility to validate its participant ids through Regular Expresion. The following property in the table BDMSL_CONFIGURATION allows validating participant ids:

Example:

Property = subdomain.validation.participantIdRegex.**peppol.acc.edelivery.tech.ec.europa.eu**

Value = ^((((1234|45678|9584|9635):).*)|(*))$

Property = subdomain.validation.participantIdRegex.**generalerds.acc.edelivery.tech.ec.europa.eu**

Value = ^.*$

NOTE: The property must start with **subdomain.validation.participantIdRegex.** followed by the existent subdomain name.

11.2. Logical Address validation per Domain

Two addresses are needed to create a SMP: the Logical and the Physical Addresses. As from SML version 3.1, the configuration allows to specify if the Logical Address may accept **HTTP** or **HTTPS** protocol for the Create SMP Operation.

An additional property has been introduced in the table BDMSL_CONFIGURATION in that purpose.The property must start with **subdomain.validation.smpLogicalAddressProtocolRestriction.** followed by the domain name.

The possible values for this property are (all, http or https). The option 'all' means that both protocols are accepted.

Example:

Property = subdomain.validation.smpLogicalAddressProtocolRestriction.**test.acc.edelivery.tech.ec.europa.eu**

Value = all

NOTE: The property must start with **subdomain.validation.smpLogicalAddressProtocolRestriction.** followed by the existent subdomain name.

12. SECURITY

12.1. DNS

12.1.1. DNS specifications

The SML specification [REF1] states in the chapter *5. DNS spoof mitigation*:

"The regular lookup of the address of the SMP for a given participant ID is performed using a standard DNS lookup. There is a potential vulnerability of this process if there exists at least one "rogue" certificate (e.g. stolen or otherwise illegally obtained). In this vulnerability, someone possessing such a rogue certificate could perform a DNS poisoning or a man-in-the-middle attack to fool senders of documents into making a lookup for a specific identifier in a malicious SMP (that uses the rogue certificate), effectively routing all messages intended for one or more recipients to a malicious access point. This attack could be used for disrupting message flow for those recipients, or for gaining access to confidential information in these messages (if the messages were not separately encrypted). One mitigation for this kind of attack on the DNS lookup process is to use DNSSEC rather than plain DNS. DNSSEC allow the authenticity of the DNS resolutions to be checked by means of a trust anchor in the domain chain. Therefore, it is recommended that an SML instance uses the DNSSEC infrastructure."

Thus, in order to mitigate the risk of DNS spoofing, the DNSSEC can be used in the eDelivery BDMSL application. The Domain Name System Security Extensions (DNSSEC) is a suite of Internet Engineering Task Force (IETF) specifications for securing certain kinds of information provided by the Domain Name System (DNS) as used on Internet Protocol (IP) networks.

3 properties allow the administrator to configure the DNSSEC:

Property	Description
dnsClient.SIG0Enabled	'true' if the SIG0 signing is enabled. Required fr DNSSEC. Possible values: true/false
dnsClient.SIG0PublicKeyName	The public key name of the SIG0 key
dnsClient.SIG0KeyFileName	The actual SIG0 key file. Should be just the filename if the file is in the classpath or in the 'configurationDir'

Table 5 - DNS Properties

Remark: It is important to be aware that the BDMSL deployed at the European Commission is not configured to use DNSSEC on the actual public DNS server:

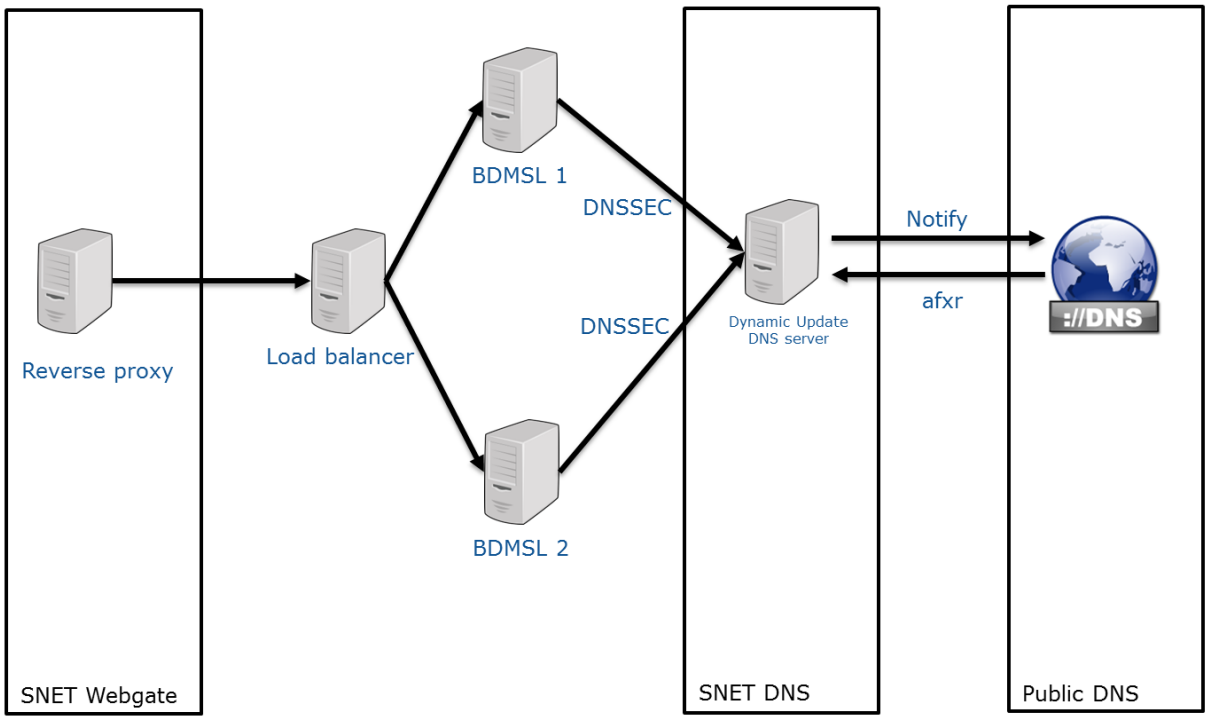

Figure 23 - BDMSL hosting at the EC

12.1.2. DNS implementation

The BDMSL registers 1 CNAME record for each SMP.

The BDMSL registers 2 types of DNS records for each participant:

- 1 CNAME record with the prefix "B-"
- 1 U-NAPTR record without prefix "B-"

Thus, for each participant, 2 records exist at the same time in the DNS and don't conflict because they don't use the same hash algorithm. For example, if a SMP registers the participant "0010:5798000000001" then:

- The MD5 hash is "e49b223851f6e97cbfce4f72c3402aac"
- The SHA-256 Base32 hash is "XUKHFQABQZIKI3YKVR2FHR4SNFA3PF5VPQ6K4TONV3LMVSY5ARVQ"

As a result, the BDMSL registers 2 records in the DNS:

```
>dig CNAME B-e49b223851f6e97cbfce4f72c3402aac.iso6523-actorid-
upis.acc.edelivery.tech.ec.europa.eu @ddnsext.tech.ec.europa.eu

B-e49b223851f6e97cbfce4f72c3402aac.iso6523-actorid-upis.edelivery.eu. 60 IN CNAME
smp.edelivery.tech.ec.europa.eu

>dig NAPTR XUKHFQABQZIKI3YKVR2FHR4SNFA3PF5VPQ6K4TONV3LMVSY5ARVQ.iso6523-actorid-
upis.edelivery.eu @ddnsext.tech.ec.europa.eu

XUKHFQABQZIKI3YKVR2FHR4SNFA3PF5VPQ6K4TONV3LMVSY5ARVQ.iso6523-actorid-
upis.edelivery.eu. 60 IN NAPTR 100 10 "U" "Meta:SMP"
"!^.*$!http://smp.edelivery.eu/iso6523-actorid-upis::0010:5798000000001!" .
```

In order to mitigate the risk of DNS spoofing, the BDMSL can use the DNSSEC infrastructure. The deployment infrastructure is described in section *12.1 DNS*.

12.2. Authentication

The authentication relies on the use of a Public Key Infrastructure (PKI). The services are all secured at the transport level with a two-way SSL / TLS connection. The requestor must authenticate using a client certificate issued for use in the infrastructure by a trusted third-party. The server will reject SSL clients that do not authenticate with a certificate issued under a trusted root.

WS-Security is only used for signing the response from the BDMSL to the SMP. It allows the SMP to validate that the request was correctly processed and acknowledged by the BDMSL.

The authentication for the user interface is also performed with 2-way SSL and the user must provide the SMP's certificate.

The authentication is performed through a custom interceptor named `CertificateAuthenticationInterceptor`. This interceptor is configured to intercept any incoming request in the `cxf-servlet.xml` configuration file:

```
<cxf:bus>
    <cxf:inInterceptors>
        <ref bean="certificateAuthenticationInterceptor"/>
    </cxf:inInterceptors>
    [...]
</cxf:bus>
```

The interceptor extracts the certificate information from the request and then validates it.

A certificate is valid if:

- The root issuer is trusted in the bdmsl_certificate_domain table.
- It is not revoked according to its certificate revocation list (CRL)
- It is valid for the current date

This certificate is then automatically used to authenticate the client using the Spring security framework. If the certificate is valid, then the client is authenticated and the certificate details are stored in the security context. Otherwise, a `UnauthorizedFault` is thrown.

12.2.1. SSL configured on the application server

The 2-way SSL configuration can be directly set up on the application server hosting the application:

Figure 24 - SSL configured on the application server

In this type of configuration, the client certificate is passed in the request and can be intercepted in the `javax.servlet.request.X509Certificate` attribute.

12.2.2. Reverse proxy with SSL

The server can be behind a reverse proxy. In this case, 2-way SSL is set up on the reverse proxy and the application server hosting the application can use the HTTP protocol:

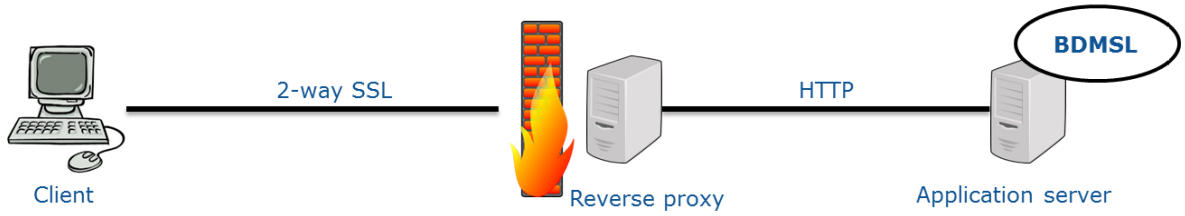

Figure 25 - Reverse proxy with SSL

In this configuration, the certificate information is stored in the HTTP header, in the `Client-Cert` attribute.

12.2.3. Admin Access

The system administrators can access the services like ChangeCertificate by authenticating through a password included in the HTTP Header in the following way:

- The HTTP header need to have the following attribute: Admin-Pwd
- The password needs to be hashed with SHA-256 Algorithm
- The password will be stored in the configuration table under the key adminPassword

12.2.4. Enable/disable BlueCoat Authentication flag

In order to authenticate into SML using the header Client-Cert attribute, the flag authentication.bluecoat.enabled must be enabled in the table BDMSL_CONFIGURATION (BlueCoat Authentications are rejected otherwise)

12.3. Authorizations

12.3.1. Roles

There are 3 roles defined in the application:

Property	Description	Current condition
ROLE_SMP	The role specific to SMP clients	The CN (Common Name) must start with "SMP_" or being a Non Root Certificate Authority and contains "_SMP_". Please see §+ 12.3.2 for further information
ROLE_PYP	The role specific to the PEPPOL Yellow Pages client	The CN starts with "PYP_"
ROLE_ADMIN	The role for the administrator of the BDMSL	No certificate needed and it needs to have the right credentials sent via the HTTP header attribute Admin-Pwd

Table 6 - Roles

Currently, the role assignment is hardcoded in the `BlueCoatClientCertificateAuthentication`,X509CertificateAuthentication `and` `AdminAuthentication` classes. For the first two, the role assignment depends on the certificate information, for the Admin it depends on the header information. This should be enhanced in a future version.

The authorizations are set using the Spring security framework using the `@PreAuthorize` annotation on the methods of the service layer:

```
@Override
@PreAuthorize("hasAnyRole('ROLE_SMP', 'ROLE_ADMIN')")
@Transactional(readOnly = false, rollbackFor = Exception.class)
public void prepareChangeCertificate(PrepareChangeCertificateBO
prepareChangeCertificateBO) throws BusinessException, TechnicalException {
    [...]
}
```

In the previous example, the method can only be called if the current client has any of the roles `ROLE_SMP` or `ROLE_ADMIN`. Otherwise, a `UnauthorizedFault` SOAP fault is thrown.

12.3.2. Granting ROLE_SMP

Root Certificate Authority

For granting a certificate as trusted and as RootCA, SML checks the *`issuer`* of the certificate against the trusted RootCA List provided by the SML database. The database flag isRootCA must be true.

A Root Certificate Authority owns a PKI (Public Key Infrastructure) to manage certificates.

Non Root Certificate Authority

For granting a certificate as trusted and as NonRootCA, SML checks the `subject` of the certificate against the trusted RootCA List provided by the SML database. The database flag isRootCA must be false.

A Non Root Certificate Authority does not own any PKI (Public Key Infrastructure) to manage certificates, a third party entity is responsible for managing certificates for such case.

Non Root and Root Certificate Priority

Apart from the aforementioned cases, SML allows certificates that are configured with Root and Non Root CA simultaneously. In such cases SML gives priority to Non Root CA, it means that if a certificate matches "Non Root CA" then SML ignores "Root CA".

NOTE: If the request method is HTTP the certificate is already trusted, for such cases SML only checks if isRootCA database flag matches.

12.4. WS-Security

If the property `signResponse` is set to true, then the responses are signed using the WS-Security framework.

The response signature is performed through a custom interceptor named `SignResponseInterceptor`. This interceptor is configured to intercept any outgoing request in the `cxf-servlet.xml` configuration file:

```
<cxf:bus>
    [...]
    <cxf:outInterceptors>
        <ref bean="signResponseInterceptor" />
    </cxf:outInterceptors>
</cxf:bus>
```

13. TECHNICAL REQUIREMENTS

This chapter describes the minimum and recommended system requirements to operate a BDMSL component.

13.1. Hardware

Type	Minimum	Recommended
Processor	1 CPU core	4 CPU core
Memory (RAM)	2GB	8GB or more
Disk space	5GB	Depends on usage

Table 7 - Hardware requirements

13.2. Software

13.2.1. Recommended stack

- Ubuntu 12.04 LTS 64 bits
- Oracle Java SE 7
- Red Hat JBoss AS 7.1.1
- MySQL 5.6
- Google Chrome

13.2.2. Operating Systems

Any operating system that is compliant with one of the supported JVM.

13.2.3. Java Virtual Machines

- Oracle Java SE JRE 7
- OpenJDK 7

13.2.4. Java Application Servers

- Apache Tomcat 8.x
- Red Hat JBoss AS 7.1.1
- Oracle WebLogic Server 12c (12.1.2.0.0)

13.2.5. Databases

- MySQL 5.6
- Oracle Database 11g (11.2.0.4.0)
- h2 1.4.190

13.2.6. Web Browsers

- Internet Explorer 8 or newer
- Mozilla Firefox
- Google Chrome

14. CONFIGURATION

14.1. Application Configuration

Property	Description
unsecureLoginAllowed	'true' if the use of HTTPS is not required. If the value is set to 'true', then the user 'unsecure-http-client' is automatically created. Possible values: true/false
configurationDir	The absolute path to the folder containing all the configuration files (truststore, keystore, sig0 key, etc.)
httpProxyHost	The http proxy host
httpProxyPort	The http proxy port
httpsProxyHost	The https proxy host
httpsProxyPort	The https proxy port
useProxy	'true' if a proxy is required to connect to the internet. Possible values: true/false
httpProxyUser	The proxy user
httpProxyPassword	The proxy password
dnsClient.enabled	'true' if registration of DNS records is required. Must be 'true' in production. Possible values: true/false
dnsClient.server	The DNS server
dnsClient.publisherPrefix	This is the prefix for the publishers (SMP). This is to be concatenated with the associated DNS domain in the table 'bdmsl_certificate_domain'
dnsClient.SIG0Enabled	'true' if the SIG0 signing is enabled. Required fr DNSSEC. Possible values: true/false
dnsClient.SIG0PublicKeyName	The public key name of the SIG0 key
dnsClient.SIG0KeyFileName	The actual SIG0 key file. Should be just the filename if the file is in the classpath or in the 'configurationDir'
signResponse	'true' if the responses must be signed. Possible values: true/false
keystoreFileName	The JKS keystore file. Should be just the filename if the file is in the classpath or in the 'configurationDir'
keystoreAlias	The alias in the keystore.
keystorePassword	The password for the keystore
paginationListRequest	Number of participants per page for the 'list' operation of 'ManageParticipantIdentifier' service. This property is used for pagination purposes.
certificateChangeCronExpression	Cron expression for the changeCertificate job. Example: 0 0 2 ? * * (every day at 2:00 am)
adminPassword	SHA-256 hashed password. Used to verify that the value provided in the HTTP header "Admin-Pwd" is correct to then attribute the role: "ROLE_ADMIN".
dnsClient.domain.*	The Value of this property represents the parent domain for this subdomain. * needs to be replaced by the subdomain, eg: dnsClient.domain**.newEntitySubdomain.edelivery.tech.ec.europa.eu**

dataInconsistencyAnalyzer.cronJobExpression	Cron expression: 0 0 3 ? * * (every day at 3:00 am)
dataInconsistencyAnalyzer.recipientEmail	Email address to receive Data Inconsistency Checker results
dataInconsistencyAnalyzer.senderEmail	Sender email address for reporting Data Inconsistency Analyzer.
subdomain.validation.smpLogicalAddressProtocolRestriction.*	The case insensitive Value of this property must be **all,http or https**. The option 'all' means that both protocols are accepted. * needs to be replaced by the subdomain, eg: subdomain.validation.smpLogicalAddressProtocolRestriction **.newEntitySubdomain.edelivery.tech.ec.europa.eu**
subdomain.validation.participantIdRegex.*	The value of this property must be populated with a regular expression to define the syntax of accepted subdomain names in addition to ISO 15459 constraints governing these identifiers. By default, the regular expression ^.*$ may be used. It accepts any sequence of characters and therefore adds no restriction. * needs to be replaced by the subdomain, eg: subdomain.validation.participantIdRegex.**newEntitySubdomain.edelivery.tech.ec.europa.eu**
authentication.bluecoat.enabled	Must be true or false. **True** means that the SML accepts and processes the header **Client-Cert.**

Table 8 - Application properties

14.2. Multiple domains

SML is able to manage DNS records per domain. Any domain must be linked to only one certificate in the database.

Domain: It is used by the SML to authenticate to the DNS server and gain update privileges.

Example: edelivery.tech.ec.europa.eu

Subdomain: Must be provided to create DNS entries for a specific domain

Example: newEntitySubdomain.**edelivery.tech.ec.europa.eu**

In order to configure domains, it is necessary to follow the steps below:

1. Define a domain in the table BDMSL_Configuration:

   Example:

- PROPERTY = **dnsClient.domain.**newEntitySubdomain.edelivery.tech.ec.europa.eu
- VALUE = edelivery.tech.ec.europa.eu

- DESCRIPTION = Domain for My Entity
- CREATED_ON = 29-MAR-17 00.00.00.000000000
- LAST_UPDATED_ON = 29-MAR-17 00.00.00.000000000

2. Define a subdomain for a specific domain in the table BDMSL_Certificate_Domain:

   Example:

- CERTIFICATE = CN=SMP_123456,O=DIGIT,C=BE
- DOMAIN = newEntitySubdomain.**edelivery.tech.ec.europa.eu**
- CRL_URL = http://crl.globalsign.net/root.crl
- CREATED_ON = 29-MAR-17 00.00.00.000000000
- LAST_UPDATED_ON = 29-MAR-17 00.00.00.000000000
- IS_ROOT_CA = 1

Note: The content of the property must starts with **"dnsClient.domain."** concatenated with the subdomain name.
The corresponding value is the domain for that specific subdomain.

14.3. Application server specific configuration

To ensure compatibility with all the supported application servers, some configuration is required.

For technical reasons, it is not possible to use the exact same war in all the application servers. Thus, it is needed to build different versions of the war. This can be done by using different maven profiles:

- `weblogic-oracle`: use this profile to build a war pre-configured for Weblogic and Oracle database.
- `tomcat-mysql`: use this profile to build a war pre-configured for Tomcat and MySQL database.
- `jboss-mysql`: use this profile to build a war pre-configured for JBoss and MySQL database.

14.3.1. Weblogic

The file `src/main/webapp/WEB-INF/weblogic.xml` has 3 purposes in the context of the BDMSL:

- Define the context root of the application
- Specify the class loading preferences for some package names (from the weblogic libraries or from the war)
- Configure the work manager to optimize the performance of the application

14.3.2. Tomcat

Tomcat is not an application server because it only supports the servlet API (including JSP, JSTL). An application server supports the whole JavaEE stack.

The file `src/main/webapp/META-INF/context.xml` has 2 purposes:

- Define the context root of the application
- Link the datasource to the globally defined JNDI datasource

14.3.3. JBoss

The file `src/main/webapp/ WEB-INF/jboss-web.xml` is responsible for defining the context root of the application.

15. ANNEXE 1 – DOCUMENT PARTS

16. LIST OF FIGURES

17. CONTACT INFORMATION

CEF Support Team

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

By phone: +32 2 299 09 09

- Standard Service: 8am to 6pm (Normal EC working Days)
- Standby Service*: 6pm to 8am (Commission and Public Holidays, Weekends)

* *Only for critical and urgent incidents and only by phone*