ELSAG DATAMAT
A Finmeccanica Company

P50638/DSASGT-0083-10/00
22 JANUARY, 2010

CONTRACT N. ESA/ESRIN 22508/09/I-LG

OPGW SOFTWARE DESIGN DOCUMENT FOR HMA FOLLOW ON TASK 4 – ORDER

SUMMARY

This document provides description of the software architectural design, the software detailed design and the internal interface design of the Ordering & Programming Gateway (OPGW) component that is in charge of providing the implementation of the Order Server required in HMA Follow On Task 4 – Order.

Prepared by:

Stefania PAPPAGALLO

(Technical Leader)

Certifies that the current document was prepared analyzing input documents and standards applicable to the project; cooperated with reference roles (OBS) reporting to a technical responsible.

Verified by:

Daniele MARCHIONNI

(Project Manager)

Certifies: consistency and completeness of the content with respect to contract and company/divisional procedures and other outputs generated by previous phases; the correspondence to applicable standards; absence of orthographic and typographical errors.

Technical Approval:

Daniele MARCHIONNI

(Project Manager)

Certifies validity of technical decisions and document coherence with both upper-level documentation and documents correlated to the expected project phase.

Quality Approval:

Raffaele BARBATI

(SPA Manager)

Certifies that: document complies with defined requirements; the process which generated it was performed as foreseen; quality records exist for the quality activities performed; any action generated was closed.

Authorized by:

Mario FONTI

(SW Engineering Manager)

Certifies that the document complies with directives and constraints set by Purchaser and that any possible commitment arisen by document is compatible with project constraints; authorizes usage and distribution.

Distribution

Pier Giorgio MARCHETTI – ESA
Yves COENE – Spacebel
Uwe VOGES – con terra

Release and Edition Register

Edition	Release	Date	Amendments description	Author
1	-		First Issue	S. Pappagallo

TABLE OF CONTENTS

1 INTRODUCTION 10
2 APPLICABLE AND REFERENCE DOCUMENTS 11
3 TERMS, DEFINITION AND ABBREVIATIONS TERMS 13
4 SOFTWARE DESIGN OVERVIEW 15
4.1 Software static architecture 15
4.1.1 Components Overview 15
4.1.2 Relationship with other systems 16
4.1.3 Information Model Overview 17
4.1.3.1 Order Options Model 17
4.1.3.2 Order Model 19
4.1.3.3 User Profile Overview 21
4.2 Software dynamic architecture 21
4.3 Interfaces Context 22
4.3.1 OPGW vs. HMA Client (Input Interfaces) 24
4.3.1.1 HMA Ordering::GetCapabilities [Discrete flow] 24
4.3.1.2 HMA Ordering::GetOptions [Discrete flow] 25
4.3.1.3 HMA Ordering::Submit [Discrete flow] 25
4.3.1.4 HMA Ordering::GetStatus [Discrete flow] 25
4.3.1.5 HMA Ordering::Cancel [Discrete flow] 26
4.3.1.6 HMA Ordering::GetQuotation [Discrete flow] 26
4.3.1.7 HMA Ordering::DescribeResultAccess [Discrete flow] 26
4.3.1.8 key exchange [Control flow] 27
4.3.2 OPGW vs. HMA Client (Output Interfaces) 27
4.3.2.1 HMA Ordering::SubmitResponse [Discrete flow] 27
4.3.2.2 HMA Ordering::CancelResponse [Discrete flow] 28
4.3.2.3 HMA Ordering::GetQuotationResponse [Discrete flow] 28
4.3.3 OPGW vs. EOLI XML Catalogue (Input Interfaces) 28
4.3.4 OPGW vs. EOLI XML Catalogue (Output Interfaces) 28
4.3.4.1 EOLI XML Catalogue::processPresentationRequest [Discrete flow] 28
4.3.5 OPGW vs. EOLI XML Order Server (Input Interfaces) 29
4.3.6 OPGW vs. EOLI XML Order Server (Output Interfaces) 29
4.3.6.1 EOLI XML Order::processProductOrderRequest [Discrete flow] 29
4.3.6.2 EOLI XML Order::processOrderMonitorRequest [Discrete flow] 29
4.3.7 OPGW vs. OPGW Operator (Input Interfaces) 29

4.3.7.1 ServiceDirectory.xml [Discrete flow] 29
4.3.7.2 Users_YYYYMMDD_HHMMSS.xml [Discrete flow] 30
4.4 Long lifetime software 30
4.5 Memory and CPU budget 30
4.6 Design Standards, conventions and procedures 31
4.6.1 UML Notations 31
4.6.1.1 Deployment Diagrams Notations 31
4.6.1.2 Component Diagrams Notations 32
4.6.1.3 Class Diagrams Notations 33
4.6.1.4 Sequence Diagrams Notations 37
4.6.2 Data Flow Diagrams Notations 37
4.6.3 Database Schema Notations 38
5 SOFTWARE DESIGN 39
5.1 General 39
5.2 Overall architecture 39
5.2.1 Components overview 39
5.3 Software components design - General 41
5.4 Software components design – Aspect of each component 42
5.4.1 OPGW - HMA Skeleton [RD-02] 42
5.4.1.1 Type 42
5.4.1.2 Purpose 42
5.4.1.3 Function 42
5.4.1.4 Subordinates 43
5.4.1.5 Dependencies 43
5.4.1.6 Interfaces 43
5.4.1.7 Resources 43
5.4.1.8 References 43
5.4.1.9 Processing 43
5.4.1.10 Data 43
5.4.2 OPGW - Order Service 44
5.4.2.1 Type 48
5.4.2.2 Purpose 48
5.4.2.3 Function 48
5.4.2.4 Subordinates 49
5.4.2.5 Dependencies 49
5.4.2.6 Interfaces 49

5.4.2.7 Resources 49
5.4.2.8 References 49
5.4.2.9 Processing 50
5.4.2.10 Data 50
5.4.3 OPGW - Support Tools 51
5.4.3.1 Type 52
5.4.3.2 Purpose 52
5.4.3.3 Function 52
5.4.3.4 Subordinates 52
5.4.3.5 Dependencies 53
5.4.3.6 Interfaces 53
5.4.3.7 Resources 53
5.4.3.8 References 53
5.4.3.9 Processing 53
5.4.3.10 Data 53
5.4.4 OPGW - Database 54
5.4.4.1 Type 55
5.4.4.2 Purpose 55
5.4.4.3 Function 55
5.4.4.4 Subordinates 55
5.4.4.5 Dependencies 55
5.4.4.6 Interfaces 55
5.4.4.7 Resources 56
5.4.4.8 References 56
5.4.4.9 Processing 56
5.4.4.10 Data 56
5.4.5 Security Layer 57
5.4.5.1 Type 58
5.4.5.2 Purpose 58
5.4.5.3 Function 58
5.4.5.4 Subordinates 58
5.4.5.5 Dependencies 58
5.4.5.6 Interfaces 58
5.4.5.7 Resources 58
5.4.5.8 References 58
5.4.5.9 Processing 58

5.4.5.10 Data 59
5.5 Dynamical Model 59
5.5.1 Identity Management Scenario 59
5.5.2 Product Ordering Scenario 60
5.5.3 Configuration Scenario 62
5.5.4 Asynchronous operations 62
5.6 Internal Interfaces design 63
5.6.1 activate Ordering Class 63
5.6.1.1 Type 63
5.6.1.2 Description 63
5.6.2 key 63
5.6.2.1 Type 63
5.6.2.2 Description 63
5.6.3 decrypt SAML Token 63
5.6.3.1 Type 63
5.6.3.2 Description 63
5.6.4 encrypt SAML Token 64
5.6.4.1 Type 64
5.6.4.2 Description 64
5.6.5 create signature 64
5.6.5.1 Type 64
5.6.5.2 Description 64
5.6.6 verify signature 64
5.6.6.1 Type 64
5.6.6.2 Description 64
5.6.7 Database Files 64
5.6.7.1 Type 64
5.6.7.2 Description 64
5.6.8 HMA Ordering ICD Interface 64
5.6.8.1 Type 64
5.6.8.2 Description 64
5.6.9 JDBC access 65
5.6.9.1 Type 65
5.6.9.2 Description 65
5.6.10 Order Capabilities File 65
5.6.10.1 Type 65

5.6.10.2 Description 65
5.6.11 Order Option Table 65
5.6.11.1 Type 65
5.6.11.2 Description 65
5.6.12 Read Order Database 65
5.6.12.1 Type 65
5.6.12.2 Description 65
5.6.13 Read User Database 65
5.6.13.1 Type 65
5.6.13.2 Description 65
5.6.14 Update Order Database 66
5.6.14.1 Type 66
5.6.14.2 Description 66
5.6.15 Update User Database 66
5.6.15.1 Type 66
5.6.15.2 Description 66
6 REQUIREMENTS TO DESIGN COMPONENTS TRACEABILITY 67

Index of Figures

Figure 4-2: OPGW Component Diagram. 16
Figure 4-2: Order Option Model. 18
Figure 4-3: Order Model. 20
Figure 4-5: User profile model. 21
Figure 4-5: OPGW Context Diagram. 22
Figure 4-7: Deployment Diagram Notations 32
Figure 4-7: Component Diagram Notations 33
Figure 4-9: Class Diagram Notations. 36
Figure 4-10: Sequence Diagram Notations 37
Figure 4-10: Data Flow Diagram Notations 38
Figure 4-12: Database Schemas Notations 38
Figure 5-1: OPGW First level decomposition. 40
Figure 5-2: Ordering Service Architecture 44
Figure 5-3: Support Tools Architecture. 51
Figure 5-4: Database Architecture. 54
Figure 5-5: OPGW DB Schema. 55
Figure 5-6: Security Layer Architecture. 57
Figure 5-7: General Handling of HMA requests scenario. 59
Figure 5-8: EO Products ordering scenario 61
Figure 5-9: OPGW Configuration scenario. 62

Index of Tables

Table 2-1. Applicable Documents.....11
Table 2-2: Reference Documents.....12
Table 3-1: Acronyms and definitions.....14
Table 6-1: OPGW Software Requirements vs. OPGW Software Components traceability matrix.....69
Table 6-2: OPGW Software Components vs. OPGW Software Requirements traceability matrix. ...71

1 INTRODUCTION

This document defines the architectural design of the ESA G/S Ordering & Programming Gateway – OPGW system that is the Order Server implementation required in HMA Follow On Task 4 – Order. OPGW provides the Ordering Service (OGC 06-141) via generating calls to appropriate back-end servers (EOLI XML Ordering Server).

This document will be updated during the HMA Follow-on project for taking into account the updates on HMA Ordering ICD that will be performed in the OGC SWG.

OPGW system is in charge of providing the following main functionalities:

- Checking the identity of the issuer of the requests according to the DAIL UM ICD [AD-11];
- Supporting the HMA Ordering ICD to return ordering options, to submit or cancel product orders (supporting possible async notification), to perform order monitoring.

The role of the OPGW is to anticipate the support for HMA Interfaces within the ESA User Service before the implementation within the ESA User Services operational systems.

2 APPLICABLE AND REFERENCE DOCUMENTS

The following table provide list of applicable documents:

Id.	Title	Reference	Issue	Date
[AD-01]	EARTHNET ONLINE XML FRONT-END INTERFACE CONTROL DOCUMENT	EOLI-XML-006-ICD	2.8	21 Jan 2008
[AD-02]	OGC™ Catalogue Services Specification 2.0 Extension Package for ebRIM (ISO/TS 15000-3) Application Profile: Earth Observation Products	OGC 06-131r6	0.2.4	07 May 2008
[AD-03]	Application schema for Earth Observation products	0GC 06-080r2	0.9.3	21 Jul 2008
[AD-04]	ECSS – Space engineering – Software	ECSS-E-ST-40C		06 Mar 2009
[AD-05]	Ordering Services for Earth Observation Products	OGC 06-141r2	0.9.5 draft 1	13 Nov 2009
[AD-06]	EARTHNET ONLINE XML FRONT-END: ORDER AND ON-LINE ACCESS EXTENSION INTERFACE CONTROL DOCUMENT	EOLI/Order-XML-ICD	3.4	07 Apr 2008
[AD-07]	OPGW Software Requirements Specification Document	OSME-USMP-SEDA-RS-08-1855	1.2	20 Mar 2009
[AD-08]	Proposal for HMA Follow On Task 4 – Order		1.0	13 Mar 2009
[AD-09]	Multi Mission User Information Services MMOHS/UMS XML ICD, UMS Import/Export XML ICD	UMS-MMOHS-XML-ICD	1.2	31 May 2006
[AD-10]	M2AS MMOHS IMPORT/EXPORT XML ICD	OSME-USMP-SEDA-IS-08-2059	1.1	21 Nov 2008
[AD-11]	User Management Interfaces for Earth Observation Services	OGC 07-118r1	0.0.4	30 Jun 2009
[AD-12]	SOFTWARE DEVELOPMENT PLAN FOR HMA FOLLOW ON TASK 4 – ORDER	P50638/DSASGT-2995-09/00	1.0	20 Nov 2009
[AD-13]	SOFTWARE PRODUCT ASSURANCE PLAN FOR HMA FOLLOW ON TASK 4 – ORDER	P-P50638/DSAQUD-3046-09/00	1.0	20 Nov 2009

Table 2-1. Applicable Documents

The following table provide list of reference documents:

Id.	Title	Reference	Issue	Date
[RD-01]	HSQLDB Web Site	http://www.hsqldb.org/		
[RD-02]	Prototype Operations Concept	HMA-PR-SPB-EN-0001	1.0	12 Aug 2006
[RD-03]	EOLI-SA Configuration File Interface Control Document	VEGA.EOLI-SA.ICD.042	1.17	16 Apr 2008
[RD-04]	GS implementation of the Catalogue, Ordering and Programming ICD's TN	HMA-RP-ASU-PM-0001		
[RD-05]	ESA/GS HMA Prototipe Software Specification Document	HMA-RS-DAT-EN-001	2.2	17 Oct 2007
[RD-06]	Apache XML Security	http://santuario.apache.org/Java/index.html		

Table 2-2: Reference Documents

3 TERMS, DEFINITION AND ABBREVIATIONS TERMS

Acronym	Definition
API	Application Programmer's Interface
AR	Acceptance Review
ASCII	American Standard for Code Information Interchange
ASN.1	Abstract Syntax Notation One
BNF	Backus-Naur Form
CDR	Critical Design Review
CM	Contributing Mission
COTS	Commercial off-the-shelf
DAIL	Data Access Integration Layer
DAP	Data Access Portfolio
DB	Database
EO-A	Enhanced On-line Access
EOLI	Earthnet On-line Interactive
EoliSA	Earthnet On-line Interactive and Stand-Alone Client: main user interface to the catalogue and ordering on-line services
ESA	European Space Agency
GDS	Guide Directory Server
GFE	Generic Front End (MMFI Ingestion Facility)
GIP	Gateway Interoperable Protocol
GMES	Global Monitoring for Environment and Security
GSC	GMES Space Component
GSDR	Ground Segment Design Review
GSOV	Ground Segment Operation Validation.
HMA	Heterogeneous Mission Accessibility
ICD	Interface Control Document
IDS	Inventory Data Server
IDS	Inventory Data Server
LTA	Long term Archive
M2EOS	Multi Mission Earth Observation Services
M2AS	Multi Mission authorization Service
M2BS	Multi Mission Browse Server
M2CS	Multi Mission Catalogue Server
MMFI	Multi Mission Facility Infrastructure
MMMC	Multi Mission Master Catalogue
MMOHS	Multi-Mission Order Handling System
MTA	Medium Term Archive
MUIS	Multi-Mission User information Services
N/A	Not Applicable
NRT	Near Real Time
OFS	Order Front-End Server
OMT	Object Modelling Technique
OPGW	ESA G/S Ordering and Programming Gate-Way
OR	Operational qualification Review
OSBE	Order SOAP Back End
OSFE	Order SOAP Front End
OSME	Operational Support, Maintenance and Evolution Contract

Acronym	Definition
PBS	Product and Browse Data Server
PBS	Product Browse System
PDR	Preliminary Design Review
PDS	Payload Data Segment
PFD	Product Formatting and Delivery (MMFI Facility)
POMS	Polar Orbit Mission Server
SAPS	Services Authorisation Proxy Server
SDS	Service Directory Server
SEDA	Serco / Elsag Datamat Consortium
SFE	SOAP Front End
SOAP	Simple Object Access Protocol
SWG	Standard Working Group
TBC	To Be Confirmed
TBD	To Be Defined
TBW	To Be Written
TCP	Transmission Control Protocol
UML	Unified Modelling Language
UMS	User Management System
USMP	User Services & Mission Planning
WSM	ORACLE Web Service Manager

Table 3-1: Acronyms and definitions

4 SOFTWARE DESIGN OVERVIEW

This chapter provides a general overview of the system specifying:

- The context in which the system is operating
- The background of the project and relationships with other projects
- The static and dynamic architecture
- The model of the main information items handled by the system
- The notations and methodologies followed for the design of the system.

4.1 Software static architecture

This section reports the main components of the system, the identified relationships, the different statuses in which the system is operating and the model of the main information items handled by it.

4.1.1 Components Overview

The Ordering & Programming Gateway - OPGW system is in charge of providing a reference implementation of HMA Ordering ICD [AD-05] in order to allow the clients to exercise their implementation of that protocol.

Due to the nature of the HMA Ordering specification, OPGW has to deal with:

- user identity management including encryption and digital signature: in fact the ordering concept is tightly linked with the user profile, user accounting and delivery and then these information must be managed with the suitable level of confidentiality. The way this information is transferred between client and server is specified in HMA User Identity Management ICD [AD-11].
- Asynchronous web services operations: the ordering process is a long lasting activity and then the operations of that specification are mainly asynchronous i.e. the client triggers the task, but the corresponding job is completed after some time (e.g. planning of the new data to acquire, actual sensing of the data, processing and delivery).
- Protocol translation: the HMA ordering service is actually implemented by reformatting the request in the protocol suitable for the legacy system OPGW is connected to i.e. EOLI XML Ordering protocol [AD-06].

The following component diagram (see §4.6.1 for UML notations) shows the main software components included in OPGW:

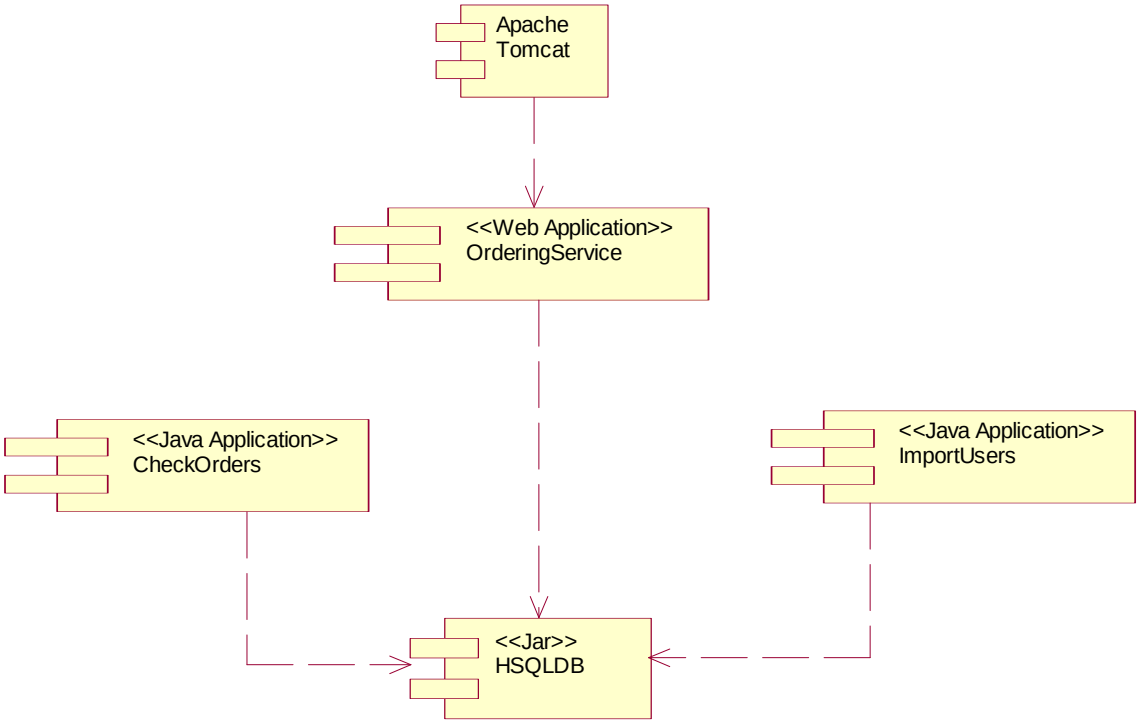

Figure 4-1: OPGW Component Diagram.

Because OPGW implements the Ordering Service ICD, which is a Web Service, then it is mainly developed as a Java Web Application deployed in Tomcat. In particular, OPGW includes the following components:

- Tomcat, which is the basic infrastructure for the implementation of the HMA Ordering Web Service
- Ordering Service, which is the java web application, developed customizing the HMA Skeleton, in charge of implementing the operations of the HMA Order Service [AD-05].
- HSQLDB, which is a light weight RDBMS written in java, that provides the storage for OPGW data i.e.: the received and submitted orders, the minimum user profile.
- ImportUsers, which is a command line Java application in charge of loading the minimal user profile information necessary for submitting an order to the EOLI XML Order server connected to OPGW.
- CheckOrders, which is a command line Java application, in charge of sending asynchronous notification to the client.

4.1.2 Relationship with other systems

OPGW has several relationships with other systems:

- **SSE**

  The SSE (Service Support Environment) is an open, interoperable system based on widely accepted standards from W3C, OASIS, WSI and OGC. It implements a Service Oriented

Architecture (SOA) facilitating access to and deployment of services and combining services using workflow technology. Services can be Ground Segment related modules such as catalogues, ordering but also external services for oil spill monitoring, fire risk, algae bloom, coordinate transformation, classification etc.

In the frame of HMA Follow-on project, SSE will be the main client of OPGW and a dedicated activity will be performed for integrating OPGW between the list of SSE connected services.

- **External OGC Catalogue**

  The product ordering process is performed after that the client has identified the EO Product of interest , then an EBRIM Catalogue has to be properly configured and loaded in order to return to users products that OPGW is able to order. Then OPGW and this catalogue shall share the configured collections and the loaded EO Product metadata.

- **M2EOS / MUIS**

  OPGW is not an ordering system (i.e. a system that does processing, formatting and delivery of products), but it is a gateway that translates the input request in a request that is suitable for triggering an actual ordering system.

  So, OPGW will be connected to M2EOS system (the new system that will replace the MUIS, which is the system currently in operation that provides access to ESA users) via the EOLI XML Order protocol [AD-06], which is the currently operationally used protocol for ordering products.

4.1.3 Information Model Overview

In order to implement the HMA Order protocol, OPGW has to store the following information items:

- ESA GS Users minimal profile information: in fact to allow the EOLI XML Order server (MUIS/M2EOS) accepting the order request, OPGW shall provide correct credentials of an actual ESA GS user, then it has to map the user identifier specified in the input request on an ESA GS user.
- Orders: because OPGW has to support asynchronous operations dealing with the status of submitted orders, and the HMA Order protocol [AD-05] is based on a stateless protocol (HTTP), then these orders must be stored in a persistent storage: for this purpose a small order database is included in OPGW.
- Order Options: for each EO Product / collection different order options are possible then this almost static information has to be held into OPGW for being returned to the client when it discovers the available ordering options.

These information items are further described in the following sub-sections.

4.1.3.1 Order Options Model

Order options is a set of combinations of allowed order option values. The issuer of the order has to choose between them and then it has to provide the needed values when submitting the order.

The picture below outlines the Order options model:

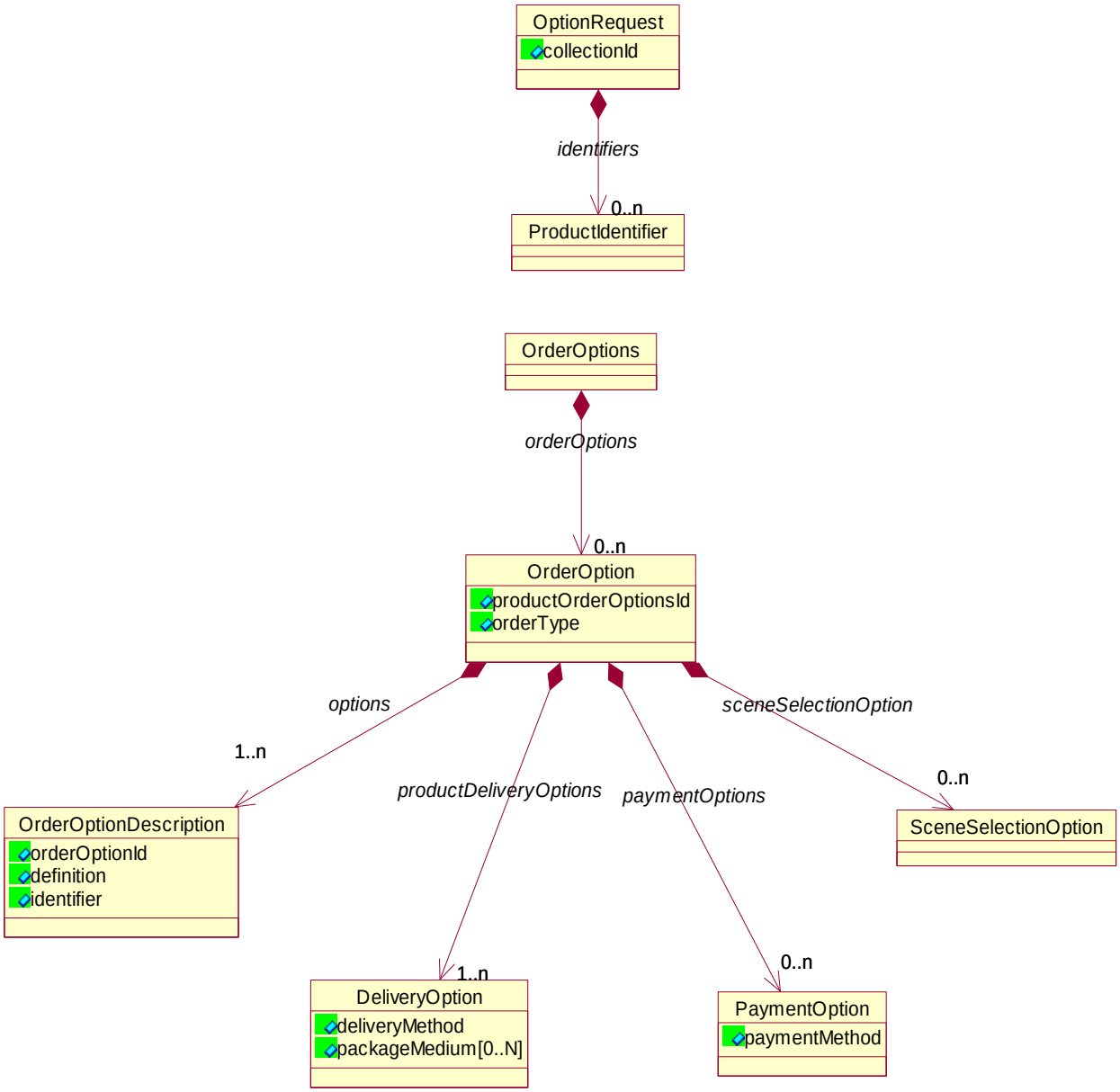

Figure 4-2: Order Option Model.

Order options are required specifying the collection id and the identifier of the product to order.

The returned order options are structured in this way:

- It is an array of order option groups (**OrderOption**)
- Each order option group has:
  - The orderOptionId: it is the name that is likely showed to the user when selecting the options for ordering from a GUI

- orderType: it specifies whether it is a product order or a subscription order
- a list of order options (**OrderOptionDescription**):
  - orderOptionId: the identifier of the option;
  - definition: it specifies whether the option is a number, or a string, or a date, etc. HMA specification allows defining every type of options (array, structures, structures with nested structures, etc.)
  - identifier: it is the identifier of the product to which the option is applicable
- productDeliveryOptions: it specifies the method (e.g. mail, electronic delivery) and the media (e.g. CD, DVD)
- paymentOptions: M2EOS supports only the "project quota" as payment.
- sceneSelectionOption: it specifies whether the product can be ordered as a strip or as a scene.

4.1.3.2 Order Model

Order is the package prepared by the user, in which the different EO products to be purchased have to be specified providing their catalogue identifier, the processing options, the delivery options and user information needed to deliver the order (delivery address).

The picture below outlines the Order model:

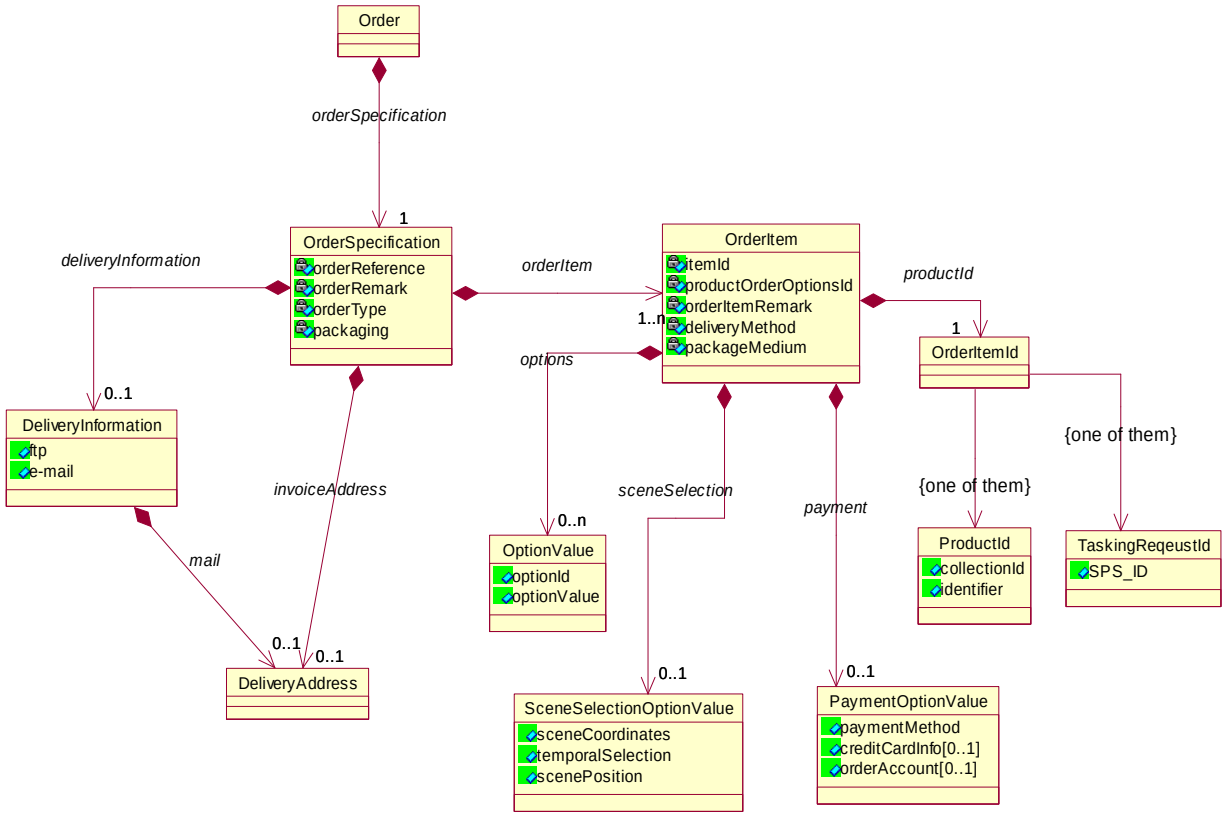

Figure 4-3: Order Model.

An order contains the following information:

- orderSpecification, which regroups the following attributes:
  - orderReference, user defined name of the order;
  - orderRemark, textual remark;
  - orderType, it specifies whether it is a product order or a subscription;
  - deliveryInformation, which include FTP / mail (with related deliver address);
  - invoice address, specifies the address where the invoice has to be put. Commercial missions are currently not supported and then the field is always empty.
  - list of orderItem, which include:
    - order item id:
      - either productId, which identifies the catalogue item to order;
      - or subscription id, which identifies the subscription to which the user is subscribing.
    - option: list of user selected options values. Several types of options can be specified: strings, numbers, enumerated strings, polygon, and times.
    - sceneSelection: in case of scene order it identifies the scene on the parent product;

- deliveryMethod & packageMedium: specifies the way and the media for delivering products.

4.1.3.3 User Profile Overview

In order to properly fill-in the EOLI XML Order sent towards the MUIS / M2EOS, OPGW needs several information about the ESA GS user the order is accounted to:

- User identifier
- Delivery Address, it is used in case the input order does not specify this information
- User Address, it is used as delivery address in case the previous information is not available
- orderAccount: it is the main information that has to be specified: it is the account to which the order assigned and it is used for updating the quota.

The following UML class diagram describes the user profile stored in OPGW.

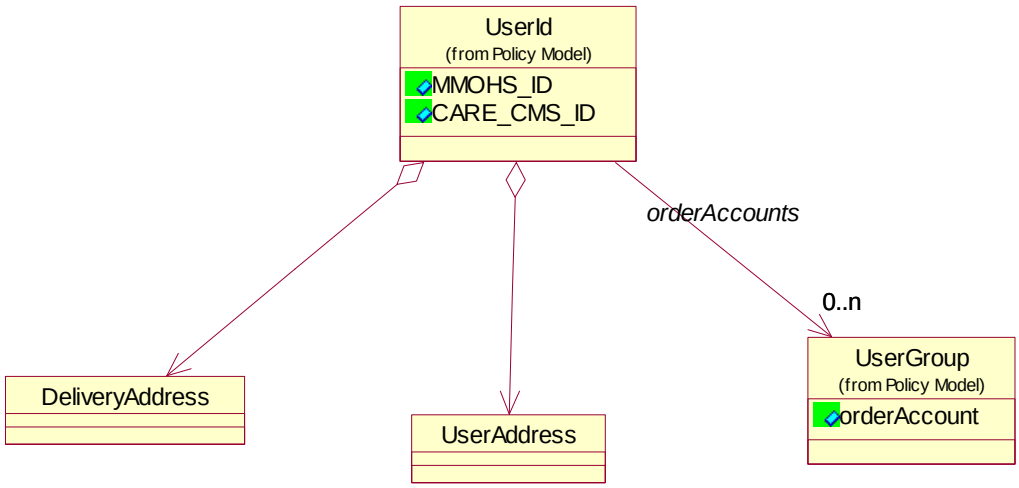

Figure 4-4: User profile model.

4.2 Software dynamic architecture

OPGW is basically the implementation of a Web Service: in fact the HMA Order protocol [AD-05] is a service composed of several operations, each implemented with SOAP messages carried by HTTP POST. Then OPGW (or at least its server part) works providing synchronous responses to synchronous requests (HTTP) and there is not a "session" between one request and another. Then this behaviour is naturally implemented developing a Web Application on an HTTP server. In particular:

- the HTTP Server is Tomcat, because it is widely used for this kind of applications and because it is used from the HMA Skeleton, which is the basis on which OPGW has been implemented.
- The web application is written in Java, as consequence of the selection of Tomcat.

However, even if the basic protocol used for carrying the messages is stateless (HTTP), the HMA Order protocol is not stateless: in fact the status of the server changes after the activation of some operations e.g.:

- when a GetQuotation is issued, the order to be quoted is stored in the system for later re-use;
- when a Submit is called, the system activates the processing, formatting and delivering of the products.

Then the status of the server must be saved and be persistent between different activation of the various Service operations. Then a DB has been included in the project. Due to the small amount of data to be managed and to the simplicity of the DB schema, then a light weight DB has been chosen: HSQLDB, an RDBMS written in java, with easy installation (just deploy a jar file) and easy usage: the DB access is done via JDBC calls.

4.3Interfaces Context

The following picture reports the context in which the OPGW will operate.

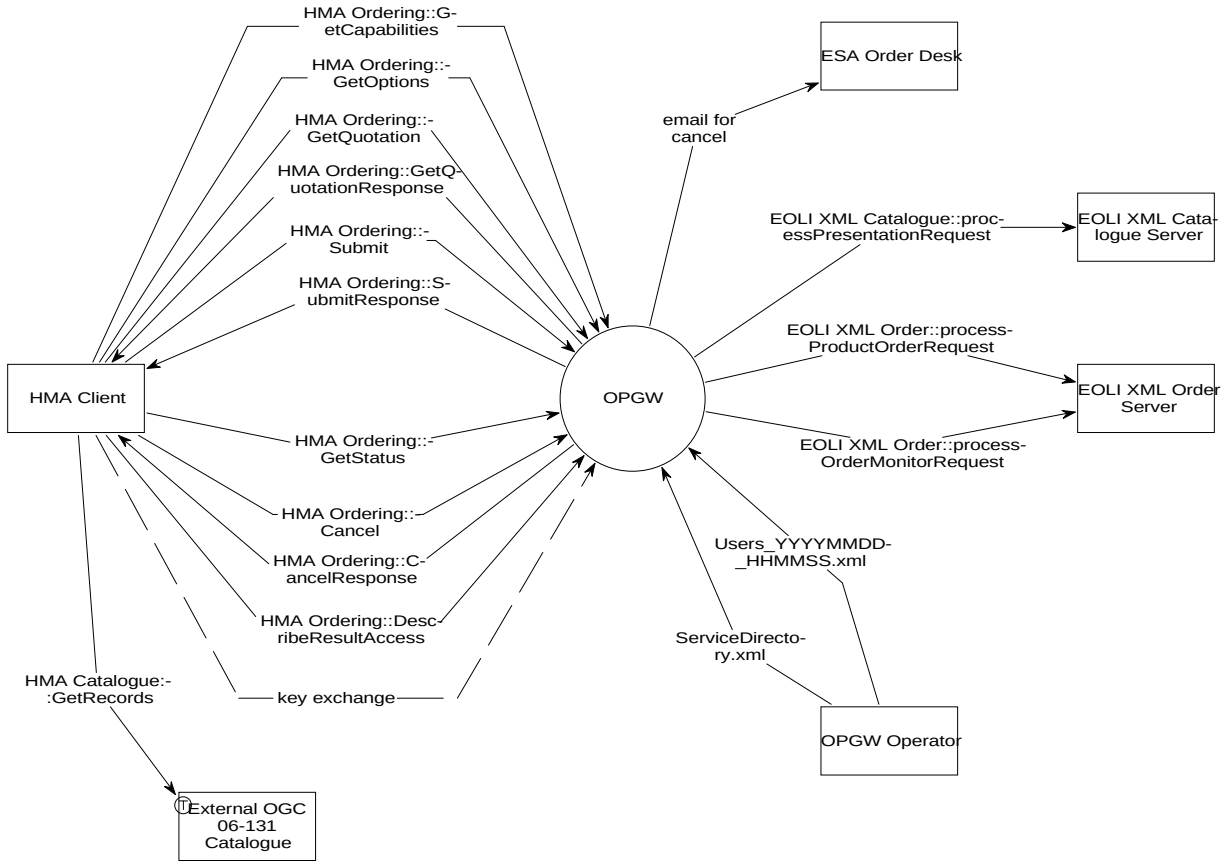

Figure 4-5: OPGW Context Diagram.

As highlighted in the previous figure, OPGW interacts with the following entities:

- **HMA Client**

  It is a client in charge of sending SOAP requests compliant with OGC 06-141. The expected HMA Clients are:

  - The TEAM engine, which will be used for validating the updated interfaces;
  - SSE, which will be connected to OPGW and added to the list of available Service Providers.

  The following are the exchanged interfaces:

  - OGC 06-131, for ordering interfaces;
  - OGC 07-118, for the encoding of user management information (it is the protocol currently used from the DAIL).

- **EOLI XML Catalogue Server**

  It is a place holder for a Web Service implementing the EOLI XML Catalogue interface for ESA collections. At the moment, this interface is provided by the MUIS system (via DSM + IDS), but MUIS is going to be replaced by M2EOS during 2010 and then OPGW will be connected to this system (actually M2AS and then M2CS sub-system).

- **EOLI XML Order Server**

  It is a place holder for a Web Service implementing the EOLI XML Ordering interface for ESA collections. At the moment, this interface is provided by MUIS system (via DSM + OFS and then MMOHS), but MUIS is going to be replaced by M2EOS during 2010 and then OPGW will be connected to this system (actually M2AS and then MMOHS).

- **OPGW Operator**

  It is the entity in charge of operating the OPGW. It has to provide the following configuration items:

  - Configuration files for the basic working of OPGW;
  - Capabilities XML files for the implemented ordering service;
  - List of configured users, with profile, i.e. the users that allowed issuing orders via OPGW.

- **External OGC 06-131 Catalogue**

  This is the OGC Catalogue storing the EO Product metadata to be queried and ordered from the HMA Client.

  The ordering of products is performed after having discovered the products of interest via a catalogue search. The catalogue is not needed for the working of OPGW: in fact it is able to process the orders on its own; the catalogue is needed for supporting the user on selecting the product of interest so OPGW has an indirect relationship with the External OGC Catalogue. The external catalogue is queried by the HMA Client for discovering the list of products available for ordering; then the HMA Client will send a product order to OPGW specifying the product identifiers returned by OGC Catalogue. Then OPGW and OGC Catalogue shall understand the same product identifiers.

  This catalogue must be kept synchronized with OPGW in order to make sure that the EO products returned by this catalogue have the same identifiers understood from OPGW and the same EO Product metadata is available also in the connected EOLI XML Catalogue Server.

- **ESA Order Desk**

The ESA order desk is notified by e-mail in case of cancellation of an HMA Order.

ESA User Services do not support the on-line cancellation of orders, but this function is supported via interaction with the Order Desk. In order to simulate the on-line cancellation of orders, OPGW will send an e-mail to a configured address asking for the cancellation of the specified order.

The interfaces exchanged between the components and entities showed in the diagram are:

- **OPGW vs. HMA Client:**
  - HMA Ordering ICD [AD-05]: it specifies the HMA interfaces for ordering products from catalogue.
  - DAIL UM ICD [AD-11]: it is not explicitly mentioned in above diagram, but it is implicitly included in all DAIL interfaces;
- **OPGW vs. EOLI XML Catalogue:**
  - EOLI XML Catalogue [AD-01]: it allows querying the EO products catalogue for getting mainly the visible products necessary for compiling an order.
- **OPGW vs. EOLI XML Order Server:**
  - EOLI XML Ordering [AD-06]: it allows issuing EO product orders.
  - Users_YYYYMMDD_HHMMSS.xml, it carries on the user profile information needed for preparing EOLI XML Orders to send.
- **OPGW vs. OPGW Operator:**
  - Monitor & control interfaces for operating, configuring OPGW.
- **HMA Client vs. External OGC Catalogue**
  - HMA Catalogue ICD [AD-02], for getting the list of available EO products.
- **OPGW vs. ESA Order Desk**
  - E-mail for cancelling an order.

The following sub-sections describes all the external interfaces showed in the context diagram above. For each interface the following information are provided:

- **Name**: identifies the interface.
- **Type**: specifies the exchange mechanism and the type of the exchanged data.
- **Description**: high level description of the interface.
- **Data Structure Description**: it provides the formal description of the exchanged data.

4.3.1 OPGW vs. HMA Client (Input Interfaces)

4.3.1.1 HMA Ordering::GetCapabilities [Discrete flow]

4.3.1.1.1 Type

SOAP message over HTTP POST (request & response)

4.3.1.1.2 Description

This message allows a client to retrieve service metadata that describe the computational and non-computational characteristics of the service.

As specified in [AD-11], the included SAML Token is encrypted using the public key of ESA GS (known also by OPGW) and then the OPGW is able to decrypt and get the needed information.

4.3.1.1.3 Data Structure Description

See [AD-05] §8.1.2

[AD-11] specifies the format of messages coming from DAIL (→ WS-security block in the SOAP header).

4.3.1.2 HMA Ordering::GetOptions [Discrete flow]

4.3.1.2.1 Type

SOAP message over HTTP POST (request & response)

4.3.1.2.2 Description

This message allows retrieval of options for ordering products from catalogue, for subscribing to a subscription, for ordering products from a linked tasking request.

As specified in [AD-11], the included SAML Token is encrypted using the public key of ESA GS (known also by OPGW) and then the OPGW is able to decrypt and get the needed information.

4.3.1.2.3 Data Structure Description

See [AD-05] §8.1.3

[AD-11] specifies the format of messages coming from DAIL (→ WS-security block in the SOAP header).

4.3.1.3 HMA Ordering::Submit [Discrete flow]

4.3.1.3.1 Type

SOAP message over HTTP POST (request & response)

4.3.1.3.2 Description

This request & response message allows submitting an order for product from a catalogue, for adhering to a subscription, for ordering products derived from a tasking request.

As specified in [AD-11], the included SAML Token is encrypted using the public key of ESA GS (known also by OPGW) and then the OPGW is able to decrypt and get the needed information.

4.3.1.3.3 Data Structure Description

See [AD-05] §8.1.6

[AD-11] specifies the format of messages coming from DAIL (→ WS-security block in the SOAP header).

4.3.1.4 HMA Ordering::GetStatus [Discrete flow]

4.3.1.4.1 Type

SOAP message over HTTP POST (request & response)

4.3.1.4.2 Description

This request & response message allows a client to retrieve the list of submitted orders, to retrieve status of precisely identified orders.

As specified in [AD-11], the included SAML Token is encrypted using the public key of ESA GS (known also by OPGW) and then the OPGW is able to decrypt and get the needed information.

4.3.1.4.3 Data Structure Description

See [AD-05] §8.1.8

[AD-11] specifies the format of messages coming from DAIL (→ WS-security block in the SOAP header).

4.3.1.5 HMA Ordering::Cancel [Discrete flow]

4.3.1.5.1 Type

SOAP message over HTTP POST (request & response)

4.3.1.5.2 Description

This request & response message allow submitting the cancellation of a already submitted order or to unsubscribe a subscription.

As specified in [AD-11], the included SAML Token is encrypted using the public key of ESA GS (known also by OPGW) and then the OPGW is able to decrypt and get the needed information.

4.3.1.5.3 Data Structure Description

See [AD-05] §8.1.10.

[AD-11] specifies the format of messages coming from DAIL (→ WS-security block in the SOAP header).

4.3.1.6 HMA Ordering::GetQuotation [Discrete flow]

4.3.1.6.1 Type

SOAP message over HTTP POST (request & response)

4.3.1.6.2 Description

This request & response message allows requesting the quotation for an order for product.

As specified in [RD-05], the included SAML Token is encrypted using the public key of ESA GS (known also by OPGW) and then the OPGW is able to decrypt and get the needed information.

4.3.1.6.3 Data Structure Description

See [AD-05] §8.1.4.

[AD-11] specifies the format of messages coming from DAIL (→ WS-security block in the SOAP header).

4.3.1.7 HMA Ordering::DescribeResultAccess [Discrete flow]

4.3.1.7.1 Type

SOAP message over HTTP POST (request & response)

4.3.1.7.2 Description

This operation allows a client to access the products ordered with on-line delivery.

As specified in [RD-05], the included SAML Token is encrypted using the public key of ESA GS (known also by OPGW) and then the OPGW is able to decrypt and get the needed information.

4.3.1.7.3 Data Structure Description

See [AD-05] §8.1.4.

[AD-11] specifies the format of messages coming from DAIL (→ WS-security block in the SOAP header).

4.3.1.8 key exchange [Control flow]

4.3.1.8.1 Type

Operator interface.

4.3.1.8.2 Description

This interface represents the exchange of the needed keys of encrypting / decrypting / signing messages.

4.3.1.8.3 Data Structure Description

See [RD-06].

4.3.2 OPGW vs. HMA Client (Output Interfaces)

4.3.2.1 HMA Ordering::SubmitResponse [Discrete flow]

4.3.2.1.1 Type

SOAP message over HTTP POST (request & response)

4.3.2.1.2 Description

This operation is called by an HMA Ordering Service for notifying the issuer of the order about the progress of the order status.

Depending on how the order has been submitted, the client can get no notification, one notification when the order reaches a final status, or all notifications.

This asynchronous notification shall include the digital signature built using the private key of ESA GS. It is needed in order to allow the DAIL to recognize the issuer of the notification.

4.3.2.1.3 Data Structure Description

See [AD-05] §8.1.7.

[AD-11] specifies the format of asynchronous answers to send to the DAIL (→ WS-security block in the SOAP header).

4.3.2.2 HMA Ordering::CancelResponse [Discrete flow]

4.3.2.2.1 Type

SOAP message over HTTP POST (request & response)

4.3.2.2.2 Description

This operation is called by an HMA Ordering Service for notifying the issuer of the cancel order.

This asynchronous notification shall include the digital signature built using the private key of ESA GS. It is needed in order to allow the DAIL to recognize the issuer of the notification.

4.3.2.2.3 Data Structure Description

See [AD-05] §8.1.11.

[AD-11] specifies the format of asynchronous answers to send to the DAIL (→ WS-security block in the SOAP header).

4.3.2.3 HMA Ordering::GetQuotationResponse [Discrete flow]

4.3.2.3.1 Type

SOAP message over HTTP POST (request & response)

4.3.2.3.2 Description

This operation is called by an HMA Ordering Service for notifying the issuer of the quotation of a specified order.

This asynchronous notification shall include the digital signature built using the private key of ESA GS. It is needed in order to allow the DAIL to recognize the issuer of the notification.

4.3.2.3.3 Data Structure Description

See [AD-05] §8.1.5.

[AD-11] specifies the format of asynchronous answers to send to the DAIL (→ WS-security block in the SOAP header).

4.3.3 OPGW vs. EOLI XML Catalogue (Input Interfaces)

None.

4.3.4 OPGW vs. EOLI XML Catalogue (Output Interfaces)

4.3.4.1 EOLI XML Catalogue::processPresentationRequest [Discrete flow]

4.3.4.1.1 Type

SOAP message over HTTP POST (request & response)

4.3.4.1.2 Description

This message allows to retrieve (multiple) product metadata within a single collection by providing the product identification.

4.3.4.1.3 Data Structure Description

See [AD-01] §2.2.2

4.3.5 OPGW vs. EOLI XML Order Server (Input Interfaces)

None.

4.3.6 OPGW vs. EOLI XML Order Server (Output Interfaces)

4.3.6.1 EOLI XML Order::processProductOrderRequest [Discrete flow]

4.3.6.1.1 Type

SOAP message over HTTP POST (request & response)

4.3.6.1.2 Description

This operation allows an EOLI XML Ordering client to issue order for products in the catalogue.

4.3.6.1.3 Data Structure Description

See [AD-06] §2.2.5, §2.2.6.

4.3.6.2 EOLI XML Order::processOrderMonitorRequest [Discrete flow]

4.3.6.2.1 Type

SOAP message over HTTP POST (request & response)

4.3.6.2.2 Description

This operation allows to monitor the order status by providing the order identifier and to retrieve all orders for a user that have been updated since a given date.

4.3.6.2.3 Data Structure Description

See [AD-06] §2.2.7, §2.2.8.

4.3.7 OPGW vs. OPGW Operator (Input Interfaces)

4.3.7.1 ServiceDirectory.xml [Discrete flow]

4.3.7.1.1 Type

XML File.

4.3.7.1.2 Description

This file carries on the order options for ESA GS Order Service.

This file is used for extracting the order options to return from GetOptions operation.

4.3.7.1.3 Data Structure Description

See [RD-03].

4.3.7.2 Users_YYYYMMDD_HHMMSS.xml [Discrete flow]

4.3.7.2.1 Type

XML File.

4.3.7.2.2 Description

This file carries on the profile information stored in EOLI XML Order Server (MMOHS). OPGW actually need only the information about user identifiers and user passwords.

4.3.7.2.3 Data Structure Description

The format is specified in [AD-09].

4.4 Long lifetime software

OPGW development is made available as open sources COTS in order to allow others to develop an order search without starting from scratch and to cope with the long planned lifetime of the software. OPGW needs of the following software resources.

- Operating System: Linux RedHat ES 5 Update 2 (being a Java development, the system can be built and operated also on Windows XP OS).
- Tomcat 6.0.18
- Ant 1.6.5
- Java 1.5.0.16;
- XML Beans 2.2.0
- HSQLDB 1.8.0.7

Using the previous listed resources it is guaranteed also the minimum dependency on the operating system to improve portability: in fact Java and the other Java based tools are available for many Operating Systems and hardware, then it is not issue to update the OS and the HW on which OPGW will be operated.

4.5 Memory and CPU budget

OPGW needs of the following hardware resources:

- CPU: at least INTEL P4 2.4 GHz or equivalent;
- RAM: at least 1GB (1 GB is needed for ORACLE COTS)
- Al least 600 MB of free disk space (to hold the OPGW & ORACLE WSM);
- Graphic Adapter: no specific needs.

4.6 Design Standards, conventions and procedures

This paragraph describes the method adopted for the architectural design of the system.

Within the project analysis and design phase, both structured and object-oriented modelling and design methods are applied:

- the first levels of the system analysis and decomposition (**Architectural Design**) are performed applying structured methods and using DFD diagrams for representation
- then the identified leaf components are further decomposed (**Detailed Design**) using object oriented methods based on UML.
- The dynamical behaviour of system is described using UML sequence diagrams.

4.6.1 UML Notations

The UML uses several kinds of models for system description. For the scope of this document the following diagrams are considered:

- **Deployment Diagrams**

  Deployment diagrams show the configuration of run-time processing elements and the software components, processes, and objects that live on them. Software component instances represent run-time manifestations of code units. Components that do not exist as run-time entities (because they have been compiled away) do not appear on these diagrams.

- **Component Diagrams**

  A component diagram shows the dependencies among software components, including source code components, binary code components, and executable components. A software module may be represented as a component type. A component diagram describes only the static relationship between software components.

- **Class Diagrams**

  Class diagrams show the static structure of the model, in particular, the things that exist (such as classes and types), their internal structure, and their relationships to other things. Class diagrams do not show temporal information, although they may contain occurrences of things that have or things that describe temporal behaviour.

- **Sequence Diagrams**

  A sequence diagram shows an interaction arranged in time sequence. In particular, it shows the objects participating in the interaction by their "lifelines" and the messages that they exchange arranged in time sequence. It does not show the associations among the objects.

In the following paragraphs the notations used in this document are explained.

4.6.1.1 Deployment Diagrams Notations

A deployment diagram shows processors, devices, and connections. Each model contains a single deployment diagram that shows the connections between its processors and devices, and the allocation of its processes to processors. The picture below shows an example of deployment diagram where a network connects two workstations, and one hosts the server process and the other hosts the client application. The white boxes are the 'Devices' of the model; the boxes with black border are the 'Processors' of the model and the straight lines are the 'Connections' of the model.

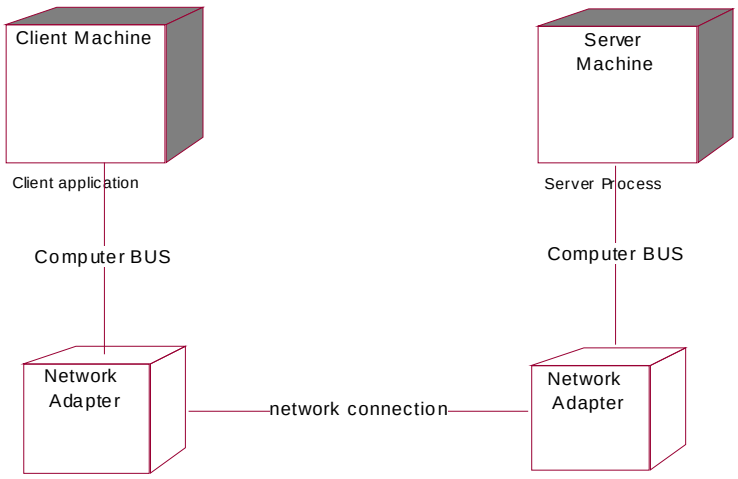

Figure 4-6: Deployment Diagram Notations

4.6.1.2 Component Diagrams Notations

Component diagrams provide a physical view of the current model. A component diagram shows the organizations and dependencies among software components, including source code components, binary code components, and executable components. These diagrams also show the externally visible behaviour of the components by displaying the interfaces of the components. Calling dependencies among components are shown as dependency relationships between components and interfaces on other components. Note that the interfaces actually belong to the logical view, but they can occur both in class diagrams and in component diagrams.

Component diagrams contain:

- **Component packages**

  Component packages represent clusters of logically related components, or major pieces of your system. Component packages parallel the role played by logical packages for class diagrams. They allow you to partition the physical model of the system.

- **Components**

  A component represents a software module (source code, binary code, executable, DLL, etc.) with a well-defined interface. The interface of a component is represented by one or several interface elements that the component provides. Components are used to show compiler and run-time dependencies, as well as interface and calling dependencies among software modules. They also show which components implement a specific class.

- **Interfaces**

  An interface specifies the externally visible operations of a class and/or component, and has no implementation of its own. An interface typically specifies only a limited part of the behaviour of a class or component.

- **Dependency relationships**

  The dependency relationship indicates that one entity in a component diagram uses the services or facilities of another.

Dependencies in the component diagram represent compilation dependencies. The dependency relationship may also be used to show calling dependencies among components, using dependency arrows from components to interfaces on other components.

In the same model one or more component diagrams are possible to depict the component packages and components at the top level of the component view, or to depict the contents of each component package. Such component diagrams belong to the component package that they depict.

The picture below illustrates the basic elements of a component diagram.

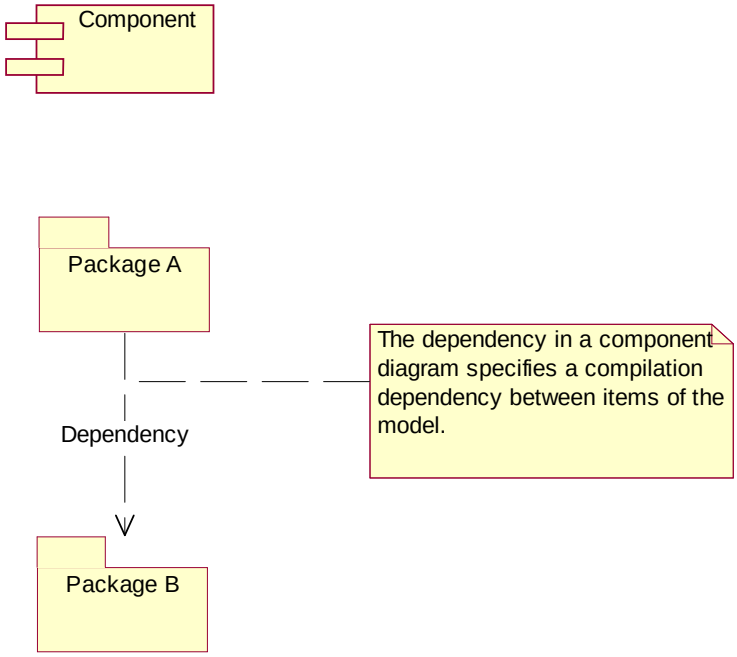

Figure 4-7: Component Diagram Notations

4.6.1.3 Class Diagrams Notations

A class diagram is a picture for describing generic descriptions of possible systems. Class diagrams and object diagrams are alternate representations of object models. Class diagrams contain classes and object diagrams contain objects, but it is possible to mix classes and objects when dealing with various kinds of metadata, so the separation is not rigid.

Class diagrams contain icons representing classes, interfaces, and their relationships. In particular, class diagrams contain:

- **Logical Packages**

  Packages serve to partition the logical model of a system. They are clusters of highly related classes that are themselves cohesive, but are loosely coupled relative to other such clusters. You can use packages to group classes, interfaces, and other packages.

- **Classes**

A class captures the common structure and common behaviour of a set of objects. A class is an abstraction of real-world items. When these items exist in the real world, they are instances of the class, and referred to as objects.

- **Interfaces**

  An interface specifies the externally visible operations of a class and/or component, and has no implementation of its own. An interface typically specifies only a limited part of the behaviour of a class or component.

- **Parameterised Classes**

  A parameterised class is a template for creating any number of instantiated classes that follow its format. A parameterised class declares formal parameters. You can use other classes, types, and constant expressions as parameters. You cannot use the parameterised class itself as a parameter. You must instantiate a parameterised class before you can create its objects.

  In its simplest form, you use parameterised classes to build container classes.

- **Instantiated Classes**

  An instantiated class is a class formed from a parameterised class by supplying actual values for parameters.

  An instantiated class is created by supplying actual values for the formal parameters of the parameterised class. This instantiation process forms a concrete class in the family of the parameterised class. The instantiated class should be put at the client end of an instantiate relationship (accessible through the Create Entry on the Tools menu) that points to the corresponding parameterised class.

- **Association Relationships**

  An association represents a semantic connection between two classes, or between a class and an interface. Associations are bi-directional; they are the most general of all relationships and the most semantically weak.

- **Aggregate Relationship**

  The aggregate relationship is used for showing a whole and part relationship between two classes.

  The class at the client end of the aggregate relationship is sometimes called the aggregate class. An instance of the aggregate class is an aggregate object. The class at the supplier end of the aggregate relationship is the part whose instances are contained or owned by the aggregate object.

  The aggregate relationship is used for showing that the aggregate object is physically constructed from other objects or that it logically contains another object. The aggregate object has ownership of its parts.

- **Generalize/Inherits Relationships**

  A generalize relationship between classes shows that the subclass shares the structure or behaviour defined in one or more super-classes. Use a generalize relationship to show an "is- a" relationship between classes.

- **Instantiates Relationships**

  An instantiates relationship represents the act of substituting actual values for the parameters of a parameterised class or parameterised class utility to create a specialized version of the more general item. In most cases, you will also draw a uses relationship

between the instantiated class and another concrete class that is used as an actual parameter.

- **Dependency Relationships**

  Draw a dependency relationship between two classes, or between a class and an interface, to show that the client class depends on the supplier class/interface to provide certain services, such as:

  - The client class accesses a value (constant or variable) defined in the supplier class/interface.
  - Operations of the client class invoke operations of the supplier class/interface.
  - Operations of the client class have signatures whose return class or arguments are instances of the supplier class/interface.

The next picture shows the items just explained.

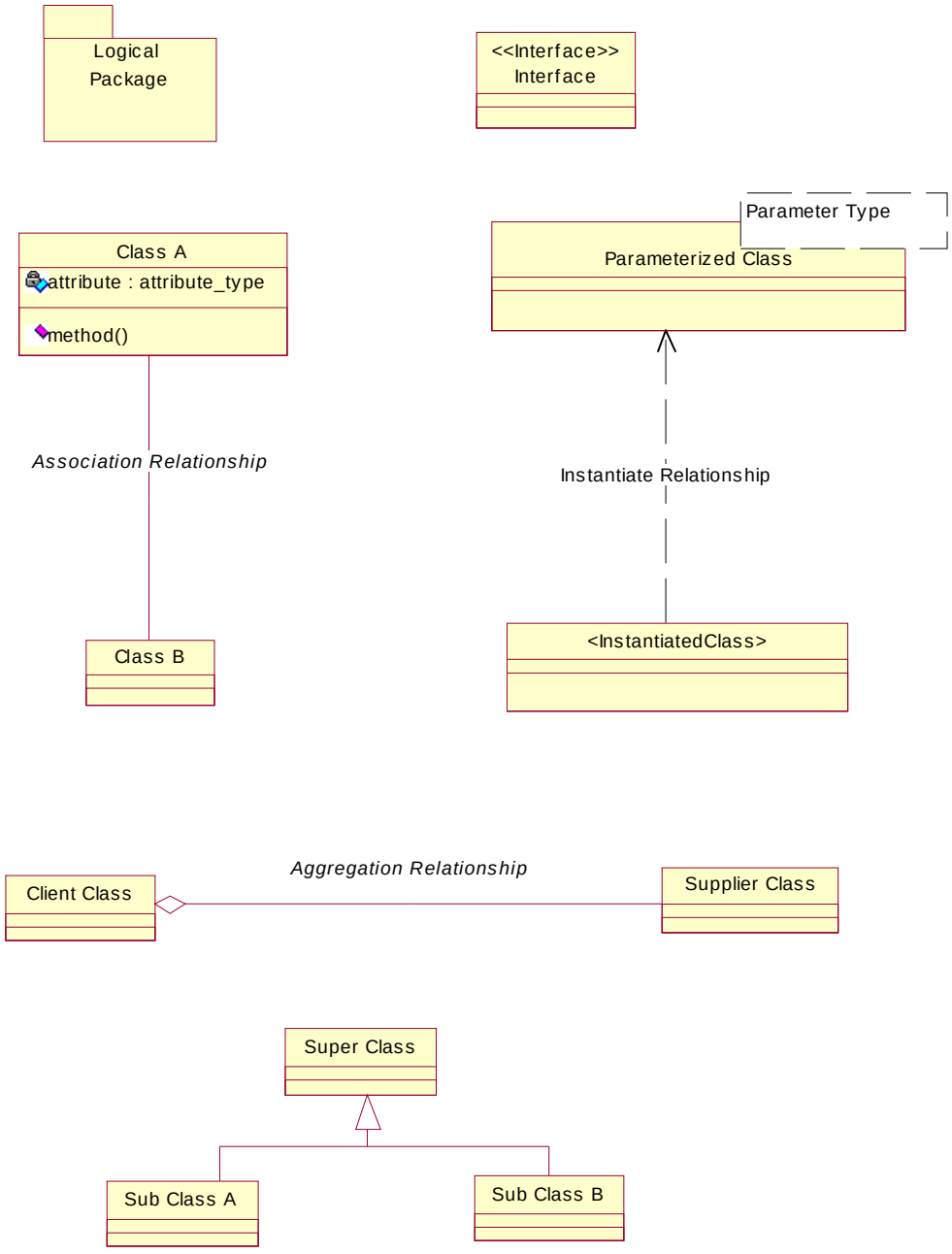

Figure 4-8: Class Diagram Notations.

4.6.1.4 Sequence Diagrams Notations

Sequence diagrams are a representation of an interaction between objects. A sequence diagram traces the execution of an interaction in time.

The picture below illustrates a sequence diagram.

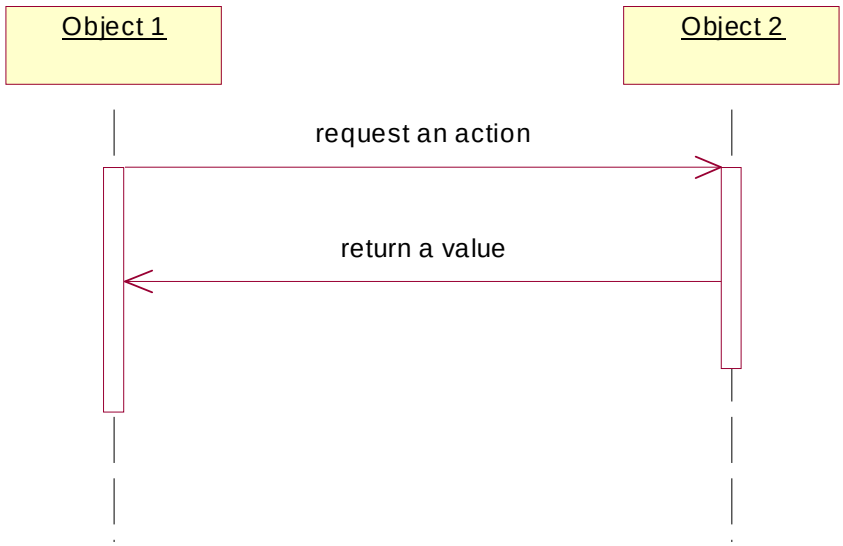

Figure 4-9: Sequence Diagram Notations

4.6.2 Data Flow Diagrams Notations

In a data flow diagram are considered the following items:

- **Processes**

  It is a function that produces or generates data. It could represent:

  - a package, that is a collection of functional homogeneous items (i.e. executables and library);
  - an executable;
  - a library;
  - a library section, that is a set of functions with an high level of cohesion within a library;
  - a function, that is a block of code of an executable devoted to make a specific task.

- **Data Flows**

  - A data flow is a flow of information exchanged between two items of the model.
  - Data Stores
  - A data store is a storage area, and is used to identify a directory of the file system, a file, and a database.

- **External Entities**

  - Is an entity external to the system to be analysed.

The picture below illustrates a data flow diagram.

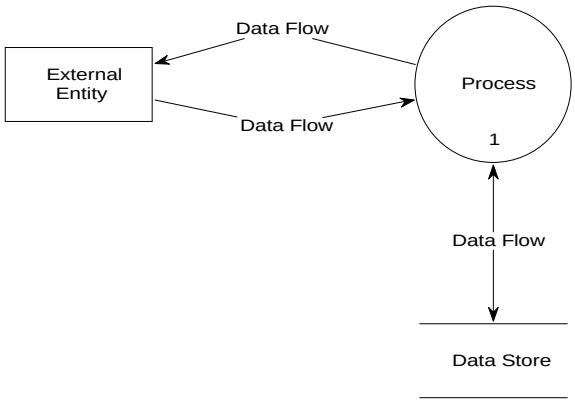

Figure 4-10: Data Flow Diagram Notations

4.6.3 Database Schema Notations

The database schemas have been represented using a notation similar to that of class diagrams. In fact relational tables have been represented with the same symbol used for classes but adding the "Relational Table" stereotype, the foreign key links between tables have been represented with the same symbol used for associations but adding the stereotype "FK".

In the following example two relational tables linked with a foreign key constraint have been represented.

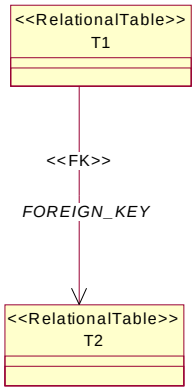

Figure 4-11: Database Schemas Notations

5 SOFTWARE DESIGN

5.1 General

This section provides the architectural decomposition of OPGW system describing:

- The different sub-components function and processing;
- The relationships and the interfaces between the different sub-components;
- The dynamical behaviour of the system.

5.2Overall architecture

5.2.1 Components overview

The following picture reports the first level decomposition of OPGW.

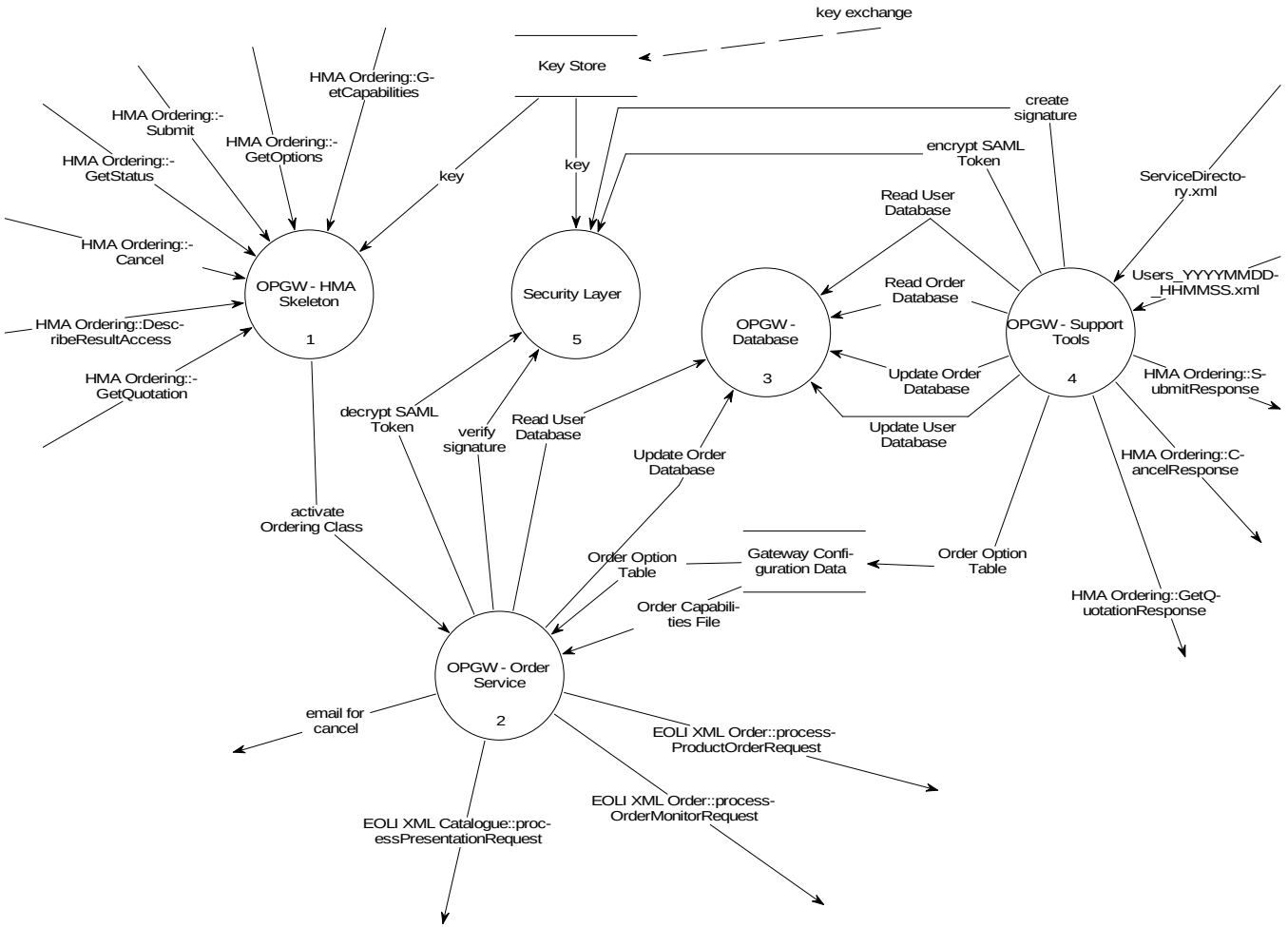

Figure 5-1: OPGW First level decomposition.

In the following sub-sections the description of OPGW level 1 components is provided.

5.3 Software components design - General

The OPGW is structured in the following sub-components:

- **OPGW - HMA Skeleton** [RD-02]

  This is the framework where the other sub-components run. It is in charge of:

  - listening incoming SOAP requests issued via HTTP protocol;
  - parse the input SOAP messages;
  - understanding the correct object to activate;

  Other special functionalities are:

  - the Skeleton can work as a server accepting requests and returning back predefined answers specified within test procedures.
  - the Skeleton includes test pages able to call the HMA Ordering interfaces provided by GSs Services.
  - The Skeleton can insert new services just by adding new java classes implementing the operations of the services to add and properly configuring the Skeleton. This capability of hosting new services by updating configuration files and deploying suitable java classes has been used for implementing OPGW services.

- **OPGW - Database**

  Package including the RDBMS (HSQLDB) and the Java library in charge of providing the functions for querying and storing permanently users and orders.

- **OPGW - Order Service**

  It is a set of java classes deployed on the Skeleton that are in charge of managing the operations defined in HMA Ordering ICD so it shall be able to manage the following operations:

  - GetCapabilities.
  - GetOptions.
  - GetQuotation.
  - Submit.
  - GetStatus.
  - Cancel.
  - DescribeResultAccess.

  Because the similarity between the HMA Ordering interfaces and the EOLI XML Order ICD interfaces, then this component works mainly translating the HMA requests in EOLI XML requests and translating back the EOLI XML response in HMA responses.

- **Support Tools**

  This is a set of Java classes providing the functionalities needed by OPGW:

  - Tool for loading ESA GS users in the OPGW database;
  - Tool for converting the order options of ServiceDirectory.xml;

- SOAP client tool for testing the system;
- Asynchronous notification: it includes Java stand alone applications for sending the notifications of asynchronous operations.

- **Security Layer**

  This is a set of java classes in charge of performing encryption, decryption, digital signature preparation and verification.

5.4 Software components design – Aspect of each component

This section provides the description of the different sub-components of OPGW system.

5.4.1 OPGW - HMA Skeleton [RD-02]

The HMA Skeleton is the environment in which the OPGW services run.

The HMA Skeleton has been developed by Spacebel in the frame of HMA-I project with the following purposes:

- Testing the HMA Client: it can work as a server accepting requests coming from the EO DAIL and returning back predefined answers specified within test procedures.
- Testing the GSs Services: the Skeleton includes test pages able to call the HMA interfaces provided by GSs Services.
- Develop GS Services: new services can be added just by adding new java classes implementing the operations of the services to add and properly configuring the Skeleton.

  This capability of hosting new services just by updating configuration files and deploying suitable java classes has been used for implementing OPGW services.

The HMA Skeleton is based on:

- Apache TOMCAT for accepting HTTP request and run Servlets;
- Apache Axis 1.x to serialize and de-serialize SOAP messages.

All the software to be deployed into the Skeleton shall be developed in Java.

5.4.1.1 Type

Sub-system (package)

5.4.1.2 Purpose

See §6.

5.4.1.3 Function

HMA Skeleton is an environment based on Java run-time, TOMCAT and Axis suitable for hosting Web Services.

It provides a basic skeleton which can be extended by updating configuration files and deploying suitable Java classes for implementing new services.

In the frame of this project it has been customized for implementing the latest version of HMA Ordering Service.

5.4.1.4 Subordinates

None.

5.4.1.5 Dependencies

The Database package shall be up and running.

5.4.1.6 Interfaces

Inputs:

- HMA Ordering::GetCapabilities [Discrete flow]
- HMA Ordering::GetOptions [Discrete flow]
- HMA Ordering::GetQuotation [Discrete flow]
- HMA Ordering::Submit [Discrete flow]
- HMA Ordering::GetStatus [Discrete flow]
- HMA Ordering::Cancel [Discrete flow]
- key [Discrete flow]
- HMA Ordering::DescribeResultAccess [Discrete flow]

Outputs:

- activate Ordering Class [Discrete flow]

5.4.1.7 Resources

See §4.4 and §4.5.

5.4.1.8 References

[RD-02]

5.4.1.9 Processing

See §5.4.1

5.4.1.10 Data

None.

5.4.2 OPGW - Order Service

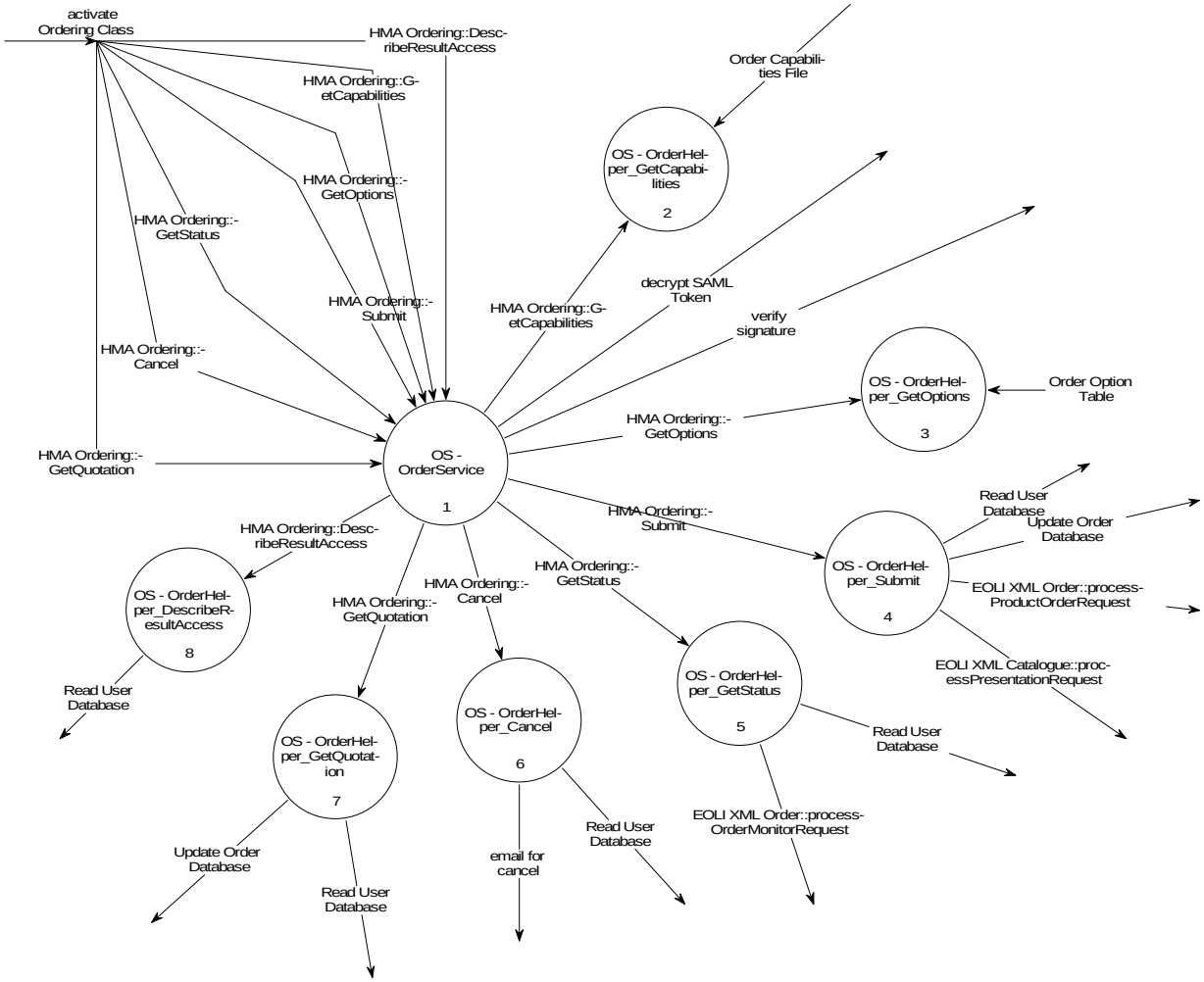

Figure 5-2: Ordering Service Architecture

The Ordering Service component is a set of Java classes in charge of implementing the operations of the HMA Ordering ICD.

It is composed of:

- **OrderService**:

  This java class is the interface between the Skeleton software and the OPGW specific software. It has to provide all the operations of the HMA Ordering ICD supported by the OPGW with the signature specified in the Skeleton documentation. Each operation is implemented by creating the corresponding Helper class and activating the corresponding operation.

- **OrderHelper_GetCapabilities:**

  This java class is in charge of providing answers to GetCapabilities requests. The response is simply built by reading an XML document containing the OPGW capabilities and returning back it within a SOAP envelope.

This operation supports also version negotiation:

It supports several capabilities files each corresponding to a specific version. Depending on the version requested by the client it returns the corresponding capabilities file or returns an error in case the version is not supported.

- **OrderHelper_GetOptions:**

This java class in charge of providing answers to GetOptions requests.

The operation is implemented accessing the Order Options table, generated by converting the EOLI SA ServiceDirectory.xml file, looking for the options matching the collection identifier specified in the input parameter.

- **OrderHelper_Submit:**

This class is in charge of managing the Submit operation of HMA Ordering ICD.

This class performs translation between HMA Submit and EOLI XML processProductOrderRequest operations following the rules specified in [AD-07].

It supports order for past products so it receives all needed information from the input request. Submit is an asynchronous request and then the service has to be able to send back notifications to the client.

The notification has to be sent if the WS-Addressing header is properly set i.e.:

- wsa:ReplyTo is not set to anonymous (http://www.w3.org/2005/08/addressing/anonymous)
- wsa:MessageID: is not empty

and if:

- statusNotification is set to "Final". As specified in [AD-07], only "None" and "Final" are the supported values.

EOLI XML Order ICD [AD-06] does not support status notification, and then the status has to be retrieved by activating order monitoring operations.

To be noted that, for building an EOLI XML Ordering request, the user name and password of the issuer of the request has to be included. Because the SAML Token embedded in Client requests does not include the password, then it has to be retrieved from the OPGW user database.

Processing:

- extraction of SOAP header:
  - WS-addressing
  - Decrypt SAML Token
  - Read the SAML Token attributes (hmaAccount)
- Parsing of the SOAP Body
- Translation of the HMA Submit in the EOLI XML processProductOrder request:
  - Set up user name and password in EOLI XML request (getting the parameters from SAML Token and user Database).
  - Map the other ordering parameters;

- Set-up the order item:
  - Set-up the resTitle starting from input identifier
  - Set-up order options from HMA ones
  - Set the order reference with a string marking that it is an order issued by OPGW and that it is related to PAST products.
  - Set up collection id

- Send the request to the EOLI XML Order Server.
- Translation of the EOLI XML response in the corresponding HMA response
- In case of asynchronous notifications, the order is stored in the order database
- A tool for managing orders that require status notification (CheckOrder tool provided by Support Tools) periodically checks the Order Database and when detects that the order has reached a final status sends a notification to the client.

– **OrderHelper_GetStatus:**

This class is in charge of managing the GetStatus operation of HMA Ordering ICD.

For HMA Ordering GetStatus and EOLI XML processOrderMonitorRequest there is almost one-one mapping. As for Submit operation, EOLI XML request shall include user name and password, then it has to be retrieved from OPGW user database.

Processing:

- extraction of SOAP header:
  - Decrypt SAML Token
- Read the SAML Token attributes (hmaAccount)
- Parsing of the SOAP Body
- Set up EOLI XML order monitoring request
- Set up user name and password in EOLI XML request (getting the parameters from SAML Token and the user Database).
- Set up the order monitoring parameters:
  - Either the order id
  - Or the filtering parameters
- Send the request to the EOLI XML Ordering server.
- Translation of the EOLI XML response in the corresponding HMA response, there is an almost one-one mapping

– **OrderHelper_Cancel:**

This class is in charge of managing the Cancel operation of HMA Ordering ICD.

It supports cancel order for past with the ID order. Cancel is an asynchronous request and then the service has to be able to send back notifications to the client.

The notification has to be sent if the WS-Addressing header is properly set i.e.:

- wsa:ReplyTo is not set to anonymous (http://www.w3.org/2005/08/addressing/anonymous)
- wsa:MessageID: is not empty

and if:

- statusNotification is set to "Final". As specified in [AD-07], only "None" and "Final" are the supported values.

As for Submit operation, the request shall include user name and password, then it has to be retrieved from OPGW user database.

Processing:

- extraction of SOAP header:
  - WS-addressing
  - Decrypt SAML Token
  - Read the SAML Token attributes (hmaAccount)
- Read the SAML Token attributes (hmaAccount)
- Parsing of the SOAP Body
- Send the request to cancel the order identified by ID to ESA Order Desk.

– **OrderHelper_DescribeResultAccess:**

This class is in charge of managing the DescribeResultAccess operation of HMA Ordering ICD. It supports the retrieving the products ordered with online delivery.

As for Submit operation, the request shall include user name and password, then it has to be retrieved from OPGW user database.

This function is not actually performed by OPGW, but it is simulated putting a test file in an appropriate directory for being downloaded.

Processing:

- extraction of SOAP header:
  - Decrypt SAML Token
- Read the SAML Token attributes (hmaAccount)
- Parsing of the SOAP Body
- Send the request to recover the order identified by ID.
- The response includes the URLs where the client can take the product ordered with online delivery and returning back it within a SOAP envelope.

– **OrderHelper_GetQuotation:**

This class is in charge of managing the GetQuotation operation of HMA Ordering ICD.

It supports quotation requests for past products so it receives all needed information from the input request. GetQuotation can be an asynchronous request and then the service has to be able to send back notifications to the client.

The notification has to be sent if the WS-Addressing header is properly set i.e.:

- wsa:ReplyTo is not set to anonymous (http://www.w3.org/2005/08/addressing/anonymous)
- wsa:MessageID: is not empty

and if:

- statusNotification is set to "Final". As specified in [AD-07], only "None" and "Final" are the supported values.

As for Submit operation, the request shall include user name and password, then it has to be retrieved from OPGW user database.

To be noted that the quotation is not supported by ESA GS and then it is simulated by OPGW just for allowing a client to test this Service operation.

Processing:

- extraction of SOAP header:
  - WS-addressing
  - Decrypt SAML Token
  - Read the SAML Token attributes (hmaAccount)
- Parsing of the SOAP Body
- In case of synchronous notification the response is the quotation and returning back it within a SOAP envelope.
- In case of asynchronous notifications, the order is stored in the order database
- A tool for managing quotation that require status notification (CheckOrder tool provided by Support Tools) periodically checks the Order Database and when it finds a quotation ready to send, send it to the client.

5.4.2.1 Type

Sub-system (java classes)

5.4.2.2 Purpose

See §6.

5.4.2.3 Function

The Ordering Service component is a set of Java classes in charge of implementing the operation of the HMA Ordering ICD:

- GetCapabilities
- GetOptions
- GetQuotation

- Submit
- GetStatus
- Cancel
- DescribeResultAccess

5.4.2.4 Subordinates

- OS - OrderHelper_GetCapabilities
- OS - OrderHelper_GetOptions
- OS - OrderHelper_Submit
- OS - OrderHelper_GetStatus
- OS - OrderHelper_Cancel
- OS - OrderHelper_GetQuotation
- OS - OrderHelper_DescribeResultAccess
- OS - OrderService

5.4.2.5 Dependencies

The Skeleton shall be up & running.

The Database shall be up & running.

5.4.2.6 Interfaces

Inputs:

- activate Ordering Class [Discrete flow]
- Order Option Table [Discrete flow]
- Order Capabilities File [Discrete flow]

Outputs:

- EOLI XML Order::processProductOrderRequest [Discrete flow]
- EOLI XML Order::processOrderMonitorRequest [Discrete flow]
- Read User Database [Discrete flow]
- Update Order Database [Discrete flow]
- EOLI XML Catalogue::processPresentationRequest [Discrete flow]
- email for cancel [Discrete flow]
- decrypt SAML Token [Discrete flow]
- verify signature [Discrete flow]

5.4.2.7 Resources

See §4.4 and §4.5.

5.4.2.8 References

None.

5.4.2.9 Processing

See §5.4.2.

5.4.2.10 Data

- Preconfigured Capabilities XML document
- Preconfigured list of order options.

5.4.3 OPGW - Support Tools

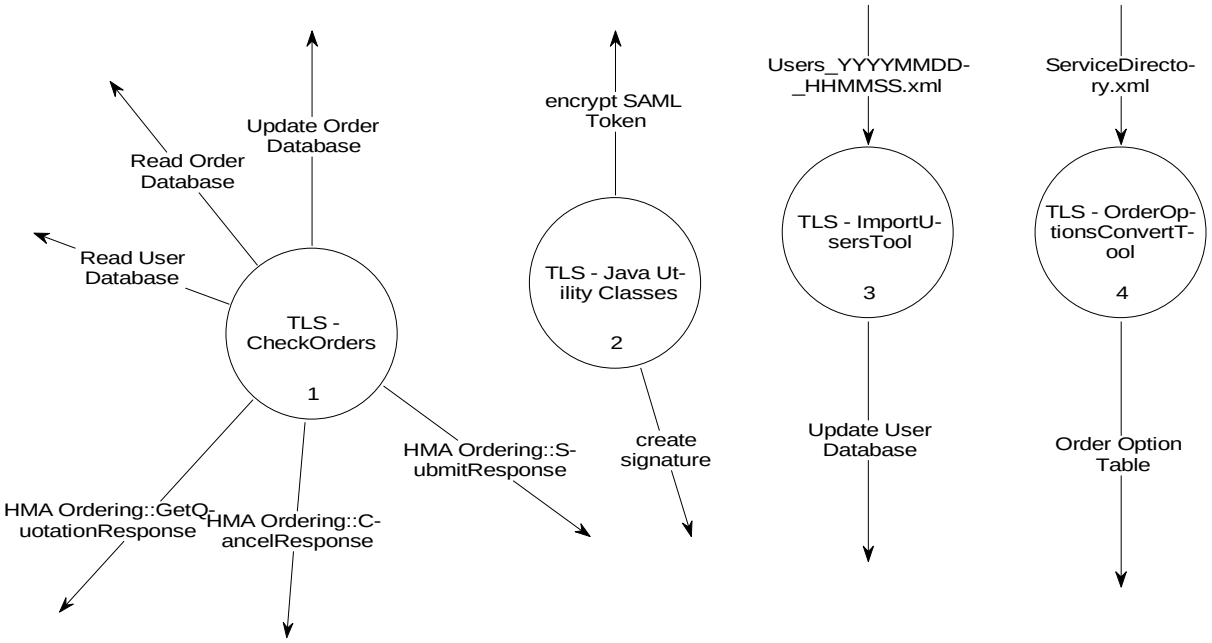

Figure 5-3: Support Tools Architecture.

This is a set of Java classes and tools providing the functionalities needed to Ordering Service component.

It includes the following sub-components:

- **CheckOrders**: stand-alone java application which is in charge of checking the progress of orders and quotation which need asynchronous notification. It scans periodically the order database table:
  - for each order of the table that require async notification, the related order status is retrieved by calling the processOrderMonitorequest operation of the EOLI XML Ordering Server. Whenever the returned status is: Cancelled, Completed, Deleted or NotValid then the notification about the completion of the order is sent to the client by calling the provided SubmitResponse operation and including the information returned by the order monitor operation response.
  - for each order of the table that is flagged for cancellation, the related order status is retrieved by calling the processOrderMonitorequest operation of the EOLI XML Ordering Server. Whenever the returned status is: Cancelled, Completed, Deleted or NotValid then the notification about the completion of the order is sent to the cient by calling the provided CancelResponse operation and including the information returned by the order monitor operation response.
  - for each order quotation found in the table, the corresponding quotation is generated by the tool and sent to the client by calling the provided GetQuotationResponse operation.

To be noted that, for sending the order monitor request the user name and password are needed, then it has to connect to the user DB for extracting relevant information.

- **Java Utility Classes:**
  - Logging of input and output messages.
  - Managing communication with the EOLI XML Oder Server (MUIS or M2EOS).
  - Managing communications with clients able to manage asynchronous notifications.
  - Handling the SAML token embedded in DAIL requests (decryption, digital signature management, extraction of SAML Token attributes).
- **ImportUsersTool**

  This is a stand alone Java application in charge of parsing the user profile defined in the following interface [AD-09]: Users_YYYYMMDD_HHMMSS.xml

  The parsed data is inserted in the DB.

  The parser of XML file is automatically generated starting from the XML schema using the XMLBeans tool. The parsed data is inserted in the DB calling the Java classes dealing with Database management hosted in the Database package.
- **OrderOptionsConverterTool:** it is a stand-alone application (python script) which is in charge of:
  - extracting the order options from the whole set of configuration parameters stored into EOLI SA ServiceDirectory.xml configuration file
  - writing the extracted data on XML file.

5.4.3.1 Type

Sub-system

5.4.3.2 Purpose

See §6.

5.4.3.3 Function

This is a set of Java classes and tools providing the functionalities needed to Order Service components. It includes the following functions:

- Asynchronous order notification
- Asynchronous quotation notification
- Asynchronous cancel notification
- Generation of order options in HMA format starting from EOLI SA XML configuration
- Ingestion of user profile information (user id and password)

5.4.3.4 Subordinates

- TLS - CheckOrders
- TLS - ImportUsersTool

- TLS - Java Utility Classes
- TLS - OrderOptionsConverterTool

5.4.3.5 Dependencies

The Database shall be up & running.

5.4.3.6 Interfaces

Inputs:

- Users_YYYYMMDD_HHMMSS.xml [Discrete flow]
- ServiceDirectory.xml [Discrete flow]

Outputs:

- HMA Ordering::SubmitResponse [Discrete flow]
- Read User Database [Discrete flow]
- Read Order Database [Discrete flow]
- Update Order Database [Discrete flow]
- Update User Database [Discrete flow]
- Order Option Table [Discrete flow]
- HMA Ordering::CancelResponse [Discrete flow]
- HMA Ordering::GetQuotationResponse [Discrete flow]
- encrypt SAML Token [Discrete flow]
- create signature [Discrete flow]

5.4.3.7 Resources

See §4.4 and §4.5.

5.4.3.8 References

None.

5.4.3.9 Processing

See §5.4.3

5.4.3.10 Data

None.

5.4.4 OPGW - Database

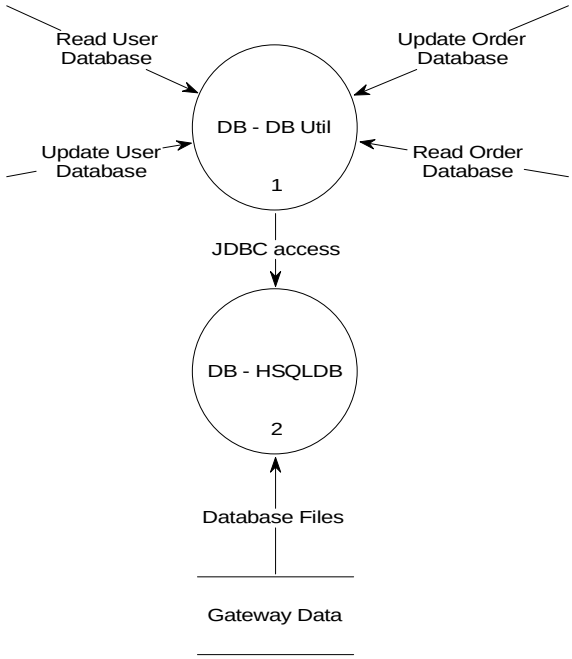

Figure 5-4: Database Architecture.

This component is in charge of providing the functions for querying and permanently storing users and orders requests.

This component includes:

- **HSQLDB**: it is the relational database engine used for managing users and orders requests.

  HSQLDB is the leading SQL relational database engine written in Java. It has a JDBC driver and supports a rich subset of ANSI-92 SQL (BNF tree format) plus SQL 99 and 2003 enhancements. It offers a small (less than 100k in one version for applets), fast database engine which offers both in-memory and disk-based tables and supports embedded and server modes. Additionally, it includes tools such as a minimal web server, in-memory query and management tools (can be run as applets) and a number of demonstration examples.

  This feature-packed software is completely free to use and distribute under licenses based on the standard BSD license. Completely free of cost or restrictions and fully compatible with all major open source licenses. Java source code and extensive documentation included.

- **DB Util:** package of Java classes encapsulating all functions for inserting, updating, reading users and orders requests. These functions are implemented uses JDBC API for accessing the DB.

DB Schema

The following figure reports the OPGW Database schema.

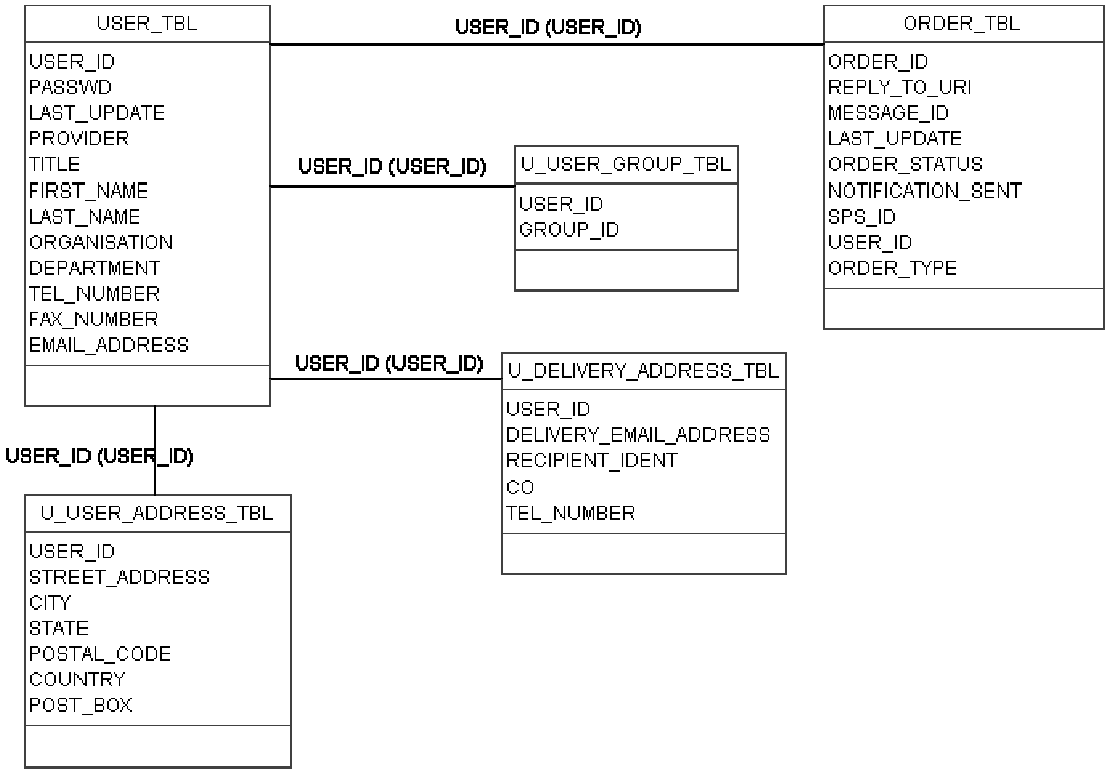

Figure 5-5: OPGW DB Schema.

5.4.4.1 Type

Sub-system (package)

5.4.4.2 Purpose

See §6.

5.4.4.3 Function

This component is in charge of providing the functions for querying and permanently storing users and orders requests.

5.4.4.4 Subordinates

- DB - DB Util
- DB - HSQLDB

5.4.4.5 Dependencies

HSQLDB shall be up and running.

5.4.4.6 Interfaces

Inputs:

- Read User Database [Discrete flow]
- Update Order Database [Discrete flow]

- Read Order Database [Discrete flow]
- Update User Database [Discrete flow]

Outputs:

- ** None **

5.4.4.7 Resources

See §4.4 and §4.5.

5.4.4.8 References

[RD-01]

5.4.4.9 Processing

See §5.4.4.

5.4.4.10 Data

This package stores:

- Several database tables for holding minimal user profile information (§5.4.4)
- A Database table (§5.4.4) needed to store HMA orders which need asynchronous notification.

5.4.5 Security Layer

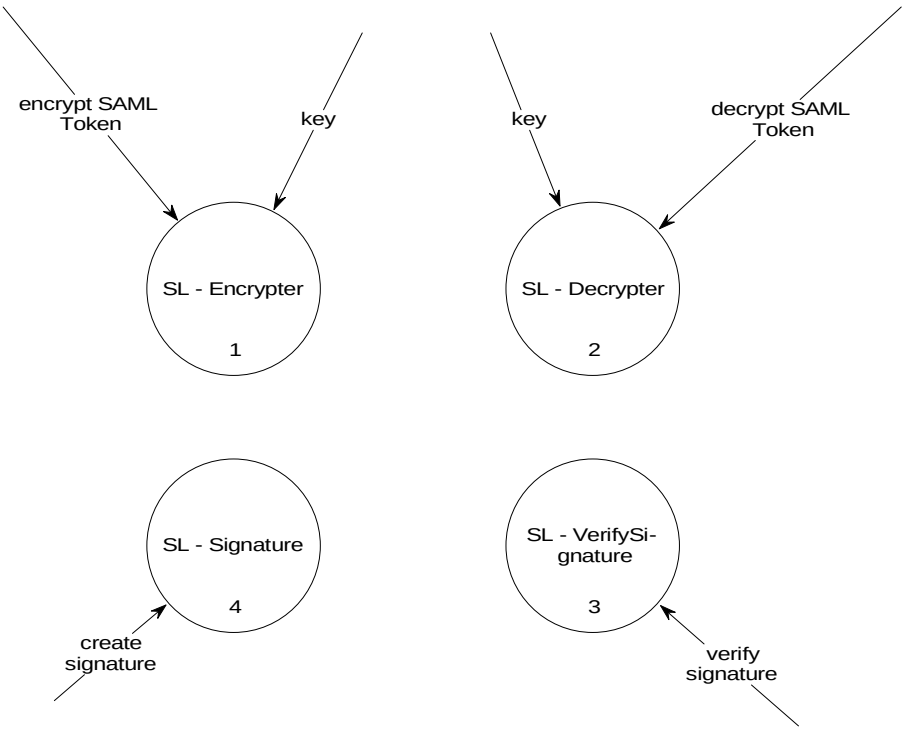

Figure 5-6: Security Layer Architecture.

This component is in charge of providing all functions related to the security of the protocol i.e.:

- SAML Token parsing
- Encryption
- Decryption
- Digital Signature Verification
- Digital Signature Preparation

This component includes:

- **SL - Signature**: it is the component used to create a digital signature to put into the response to send to the client. This module is called when an order notification has to be sent. In fact in that case the signature of OPGW must be included in the async notification.
- **SL - Encrypter**: it is the component used to encrypt the message to send to the client. This module is called when an order notification has to be sent. In fact in that case a signed and encrypted SAML Token must be included in the async notification.
- **SL - Dencrypter**: it is the component used to decrypt the message received by the client. This module is called for decrypting the SAML Token of each incoming message apart from GetCapabilitites.

- **SL - VerifySignature**: it is the component used to verify the signature included in the SAML Token of incoming requests.

These functions are implemented using the security functionalities provided by the Apache XML Security Java library [RD-06].

5.4.5.1 Type

Sub-system (package)

5.4.5.2 Purpose

See §6.

5.4.5.3 Function

This component is in charge of providing the functions for security management.

5.4.5.4 Subordinates

- SL – Encrypter
- SL - Dencrypter
- SL - Signature
- SL - VerifySignature

5.4.5.5 Dependencies

None.

5.4.5.6 Interfaces

Inputs:

- decrypt SAML Token [Discrete flow]
- encrypt SAML Token [Discrete flow]
- key [Discrete flow]
- create signature [Discrete flow]
- verify signature [Discrete flow]

Outputs:

- ** None **

5.4.5.7 Resources

See §4.4 and §4.5.

5.4.5.8 References

[RD-06]

5.4.5.9 Processing

See §5.4.5.

5.4.5.10 Data

None.

5.5Dynamical Model

This section describes the main scenarios supported by OPGW.

5.5.1 Identity Management Scenario

From [AD-11], the interaction between the HMA Client and the OPGW can be summarized in this way:

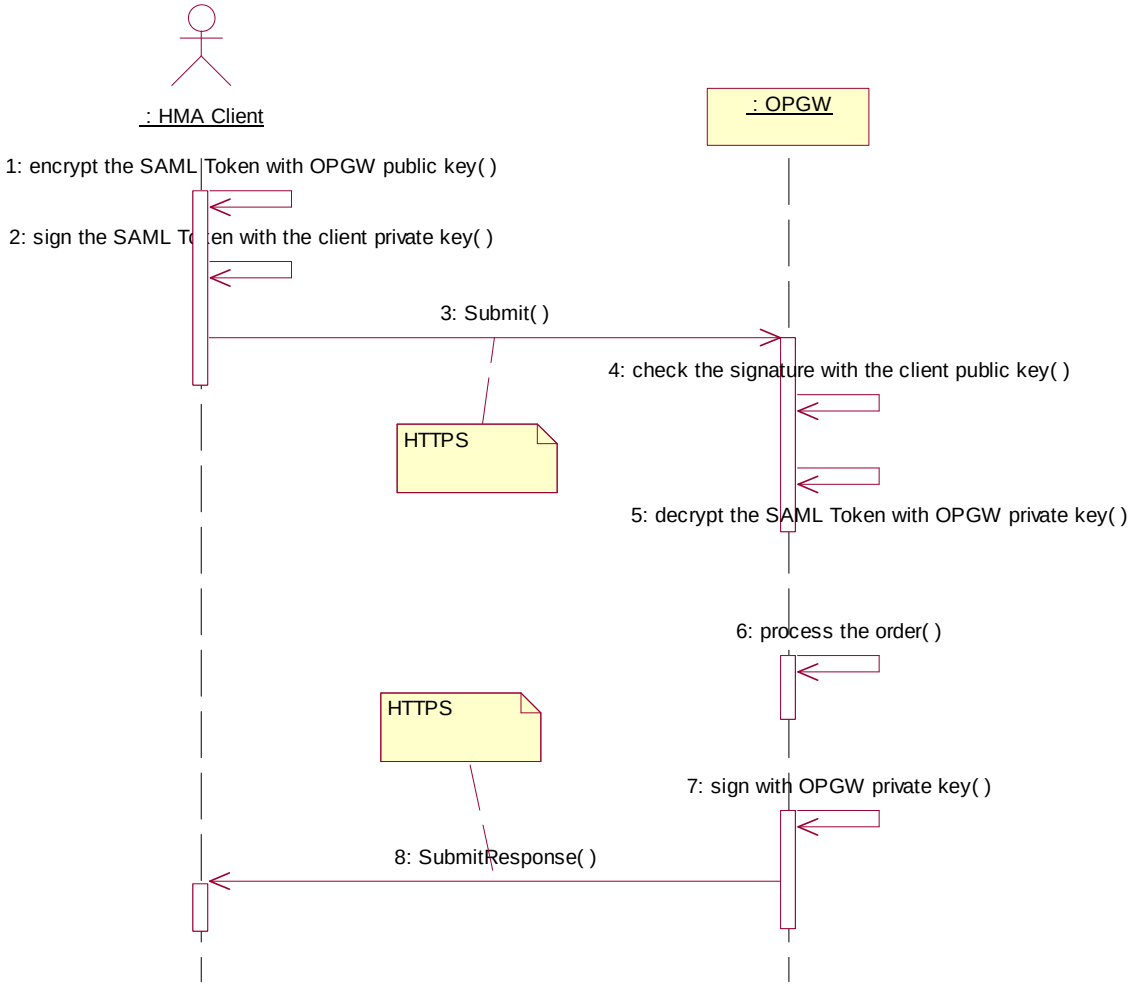

Figure 5-7: General Handling of HMA requests scenario.

- The HMA client generates a request for OPGW. The OPGW is called on **HTTPS channel**, including the **encrypted SAML Token** in the SOAP header and the service request in the

SOAP body. The **SAML Token includes the attributes of the minimum profile** [AD-11], which includes at least:

- HMA user identifier
- Country of origin
- Organisation
- Names of projects with which user is affiliated.
- The HMA account number

- The OPGW performs decryption and checking of signature, extract the minimum profile from the SAML Token, then process the request.

OPGW does not perform any authorization checks on the requests (possible restrictions are applied from the EOLI XML Order Server on the order translated from OPGW).

The second part of the scenario describes the asynchronous notification, which is possible in case of order submission. For sending the notification it is sufficient signing the message with the private key of the sender (i.e. OPGW) and sending the message via HTTPS. The DAIL, using the public key of the sender (OPGW) will check the signature: if the check is OK, then the HMA Client is sure that the sender is actually what it claims to be.

5.5.2 Product Ordering Scenario

This scenario summarizes all the interactions between HMA Client, OPGW, the companion OGC 06-131 catalogue and the EOLI XML Ordering Server (i.e. MUIS – DSM or M2EOS + MMOHS).

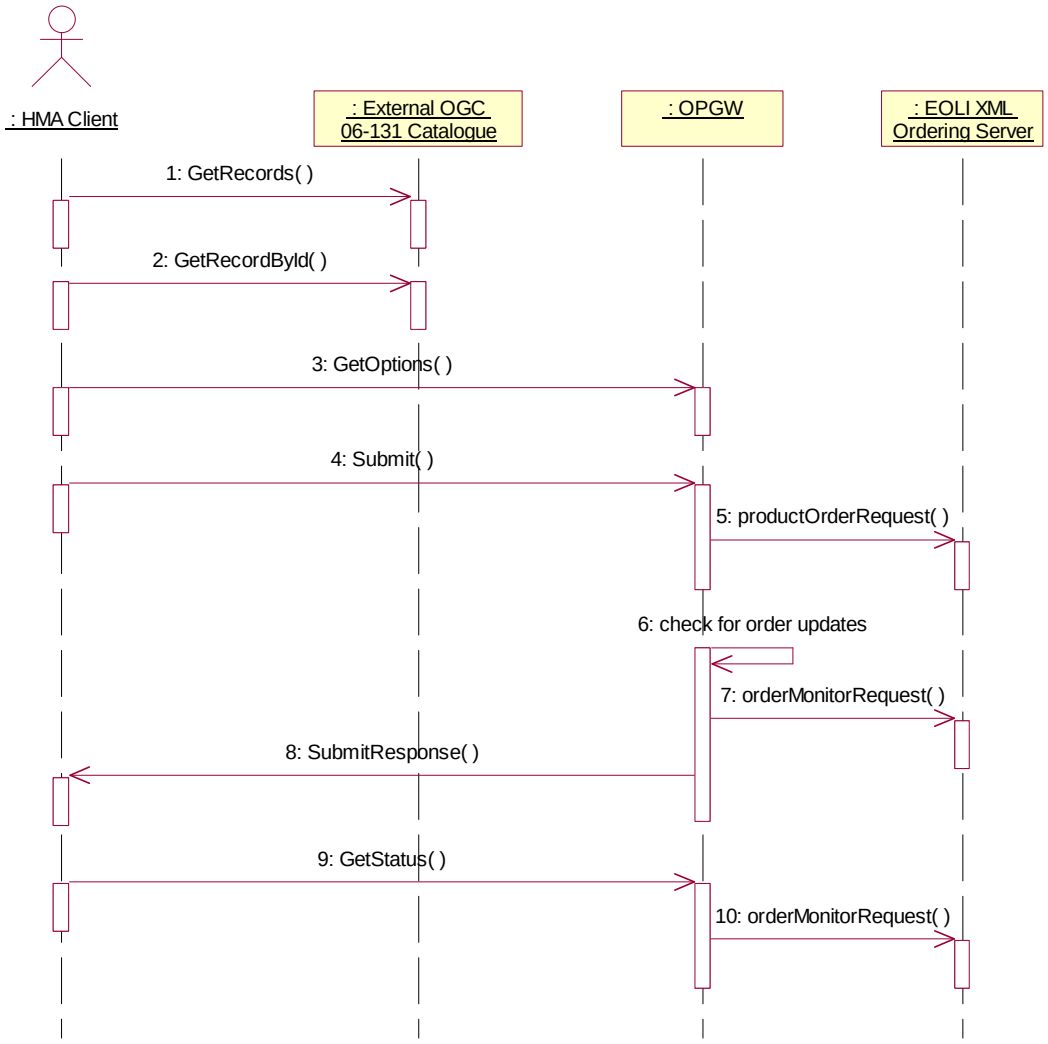

Figure 5-8: EO Products ordering scenario

This scenario does not show the steps related to identity management (see §5.5.1).

Description:

1. The HMA Client issues a query to the External OGC 06-131 Catalogue, via calling the GetRecords operation and then (optionally) GetRecordById operation.
2. Once a catalogue item has been selected, the following actions are performed for ordering the product:
   - The HMA Client calls the GetOptions on the chosen catalogue item
   - The OPGW returns the set of preconfigured options
   - From the HMA Client the user fills in all needed order options and submits the order. The HMA Client specifies to receive the notification on order completion.
   - The OPGW formats an EOLI XML Ordering request according to the input request and then send it to the EOLI XML Ordering server.

- Because EOLI XML does not support asynchronous notification, then the OPGW performs polling on the status of the submitted order waiting for the order completion.
- When the order is completed the order notification is sent to the HMA Client.
- The HMA Client can ask also the status of the order calling the GetStatus operation.

5.5.3 Configuration Scenario

This scenario describes the main information items to be configured in OPGW.

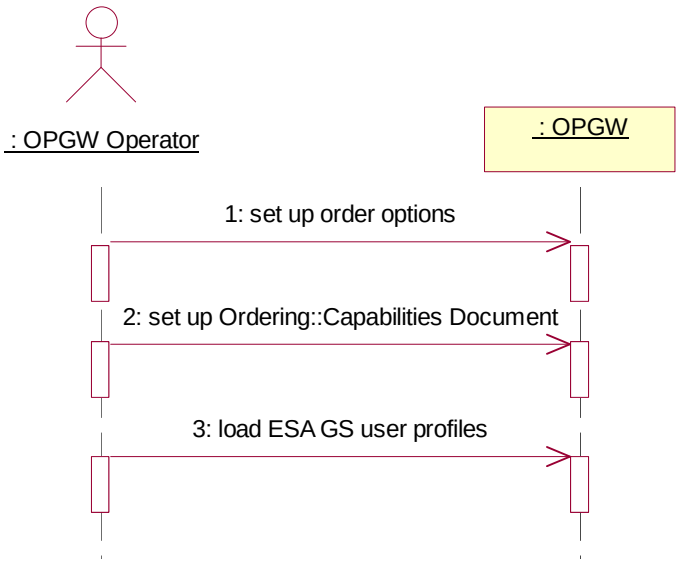

Figure 5-9: OPGW Configuration scenario.

As highlighted in the diagram:

- The answers to GetCapabilities (Capabilities documents) are preconfigured;
- Order options are preconfigured by accessing the configuration file, ServiceDirectory.xml.

5.5.4 Asynchronous operations

HMA Ordering ICD encompasses several asynchronous operations:

- GetQuotation (call back operation: GetQuotationResponse)
- Submit (call back operation: SubmitResponse)
- Cancel (call back operation: CancelResponse)

These operations work in this way:

The client calls one of these operations and it is allowed to specify whether:

- It has to receive only the acknowledge and no notifications;
- It has to receive the acknowledge and the final notifications when then processing of the operation is completed;

- It has to receive the acknowledgement and all notifications arising from the processing of the request.

When the client asks to be notified (via a specific flag of the request), then in the SOAP header it has to specify the message identifier (wsa:MessageID) and the reply address (wsa: ReplyTo/wsa:Address). Then whenever the server has to notify the client of some event it calls the corresponding <XXX>Response operation on the end point specified in the reply address.

5.6 Internal Interfaces design

In the following section all the OPGW internal interfaces highlighted in §Figure 5-1 are listed and described.

For each interface the following information are provided:

- **Name**: identifies the interface
- **Type**: specifies the exchange mechanism and the type of the exchanged data.
- **Description**: high level description of the interface.

5.6.1 activate Ordering Class

5.6.1.1 Type

Java method call.

5.6.1.2 Description

This data flow represents the calling of a Java method of the Ordering Service class when the HMA Skeleton receives a request for the HMA Ordering Service.

5.6.2 key

5.6.2.1 Type

File

5.6.2.2 Description

This file carries on the keys needed to:

- Verify the digital signature embedded from DAIL in the SAML Token
- Decrypt SAML Assertions within SAML Token sent from the DAIL.

5.6.3 decrypt SAML Token

5.6.3.1 Type

Java method call.

5.6.3.2 Description

This data flow represents the calling of a Java methods to manage decryption of SAML Token.

5.6.4 encrypt SAML Token

5.6.4.1 Type

Java method call.

5.6.4.2 Description

This data flow represents the calling of a Java methods to manage encryption of SAML Token.

5.6.5 create signature

5.6.5.1 Type

Java method call.

5.6.5.2 Description

This data flow represents the calling of a Java methods to manage creation of digital signature.

5.6.6 verify signature

5.6.6.1 Type

Java method call.

5.6.6.2 Description

This data flow represents the calling of a Java methods to manage the verification of digital signature.

5.6.7 Database Files

5.6.7.1 Type

File

5.6.7.2 Description

These are the files HSQLDB uses for storing its Database.

5.6.8 HMA Ordering ICD Interface

5.6.8.1 Type

Web Services Calls

5.6.8.2 Description

This data flow summarizes all operations of HMA Ordering ICD.

5.6.9 JDBC access

5.6.9.1 Type

JDBC calls

5.6.9.2 Description

This data flow represents the access to the HSQLDB via JDBC protocol performed by Java code using JDBC API.

5.6.10 Order Capabilities File

5.6.10.1 Type

XML File

5.6.10.2 Description

Preconfigured XML file specifying the Capabilities document for a specific version of HMA Ordering ICD.

OPGW shall have one file for each supported version.

5.6.11 Order Option Table

5.6.11.1 Type

XML File

5.6.11.2 Description

XML file including the order options for all configured products / collections.

5.6.12 Read Order Database

5.6.12.1 Type

Java method call

5.6.12.2 Description

This flow represents the call of Java methods in charge of extracting data from Order Database Tables.

5.6.13 Read User Database

5.6.13.1 Type

Java method call

5.6.13.2 Description

This flow represents the call of Java methods in charge of extracting data from USER_TBL Database Table.

5.6.14 Update Order Database

5.6.14.1 Type

Java method call

5.6.14.2 Description

This flow represents the call of Java methods in charge of inserting new orders or for modifying existing orders in the Order Database.

5.6.15 Update User Database

5.6.15.1 Type

Java method call

5.6.15.2 Description

This flow represents the call of Java methods in charge of inserting new users or for modifying existing users in the User Database.

6 REQUIREMENTS TO DESIGN COMPONENTS TRACEABILITY

SR Id	SW Component Id	Comment
SR-OPGW-FN-001	OPGW - Ordering Service	
SR-OPGW-FN-001	OPGW - Database	
SR-OPGW-FN-001	OPGW - Support Tools	
SR-OPGW-FN-005	OPGW - Ordering Service	
SR-OPGW-FN-005	OPGW - Database	
SR-OPGW-FN-005	OPGW - Support Tools	
SR-OPGW-FN-010	OPGW - Ordering Service	
SR-OPGW-FN-010	OPGW - Support Tools	
SR-OPGW-FN-020	OPGW - Database	
SR-OPGW-FN-020	OPGW - Support Tools	
SR-OPGW-FN-020	OPGW - Ordering Service	
SR-OPGW-FN-030	OPGW - Ordering Service	
SR-OPGW-FN-500	OPGW - HMA Skeleton [RD-02]	
SR-OPGW-FN-500	OPGW - Support Tools	
SR-OPGW-FN-510	OPGW - HMA Skeleton [RD-02]	
SR-OPGW-FN-510	OPGW - Support Tools	
SR-OPGW-FN-520	OPGW - Support Tools	
SR-OPGW-FN-530	OPGW - Support Tools	
SR-OPGW-FN-540	N/A	It is a requirement for the configuration of DAIL / MMOHS.
SR-OPGW-PR-001	OPGW - Ordering Service	
SR-OPGW-PR-010	OPGW - Ordering Service	
SR-OPGW-IF-001	OPGW - Ordering Service	
SR-OPGW-IF-010	OPGW - Ordering Service	

SR Id	SW Component Id	Comment
SR-OPGW-IF-020	OPGW - Support Tools	
SR-OPGW-IF-030	OPGW - Ordering Service	
SR-OPGW-IF-040	OPGW - Ordering Service	
SR-OPGW-IF-040	OPGW - Support Tools	
SR-OPGW-IF-050	OPGW - Ordering Service	
SR-OPGW-IF-060	OPGW - Order Service	
SR-OPGW-IF-070	OPGW - Order Service	
SR-OPGW-IF-070	OPGW - Database	
SR-OPGW-IF-080	OPGW - Order Service	
SR-OPGW-IF-080	OPGW - Database	
SR-OPGW-IF-080	OPGW - Support Tools	
SR-OPGW-IF-090	OPGW - Support Tools	
SR-OPGW-IF-090	OPGW - Database	
SR-OPGW-IF-100	OPGW - Order Service	
SR-OPGW-IF-110	OPGW - Order Service	
SR-OPGW-OP-001	OPGW - HMA Skeleton [RD-02]	Log files are generated by the Skeleton for each received request. The inspection of log files can be done using an ASCII (or better) an XML editor.
SR-OPGW-OP-010	All	
SR-OPGW-OP-020	OPGW - HMA Skeleton [RD-02]	Log files are generated by the Skeleton for each received request. The inspection of log files can be done using an ASCII (or better) an XML editor.
SR-OPGW-RS-001	All	
SR-OPGW-RS-010	All	
SR-OPGW-RS-020	All	
SR-OPGW-RS-030	All	

SR Id	SW Component Id	Comment
SR-OPGW-DI-001	All	
SR-OPGW-DI-010	All	
SR-OPGW-DI-020	All	PQA requirement.
SR-OPGW-SP-001	OPGW - HMA Skeleton [RD-02]	
SR-OPGW-SP-001	OPGW - Support Tools	
SR-OPGW-PB-001	OPGW - Database	
SR-OPGW-PB-001	OPGW - HMA Skeleton	
SR-OPGW-QA-001	All	PQA requirement
SR-OPGW-QA-010	All	Management Requirement
SR-OPGW-RM-001	All	
SR-OPGW-RM-010	OPGW - Ordering Service	
SR-OPGW-DD-001	OPGW - Database	

Table 6-1: OPGW Software Requirements vs. OPGW Software Components traceability matrix.

SW Component Id	SR Id	Comment
All	SR-OPGW-OP-010	
All	SR-OPGW-RS-001	
All	SR-OPGW-RS-010	
All	SR-OPGW-RS-020	
All	SR-OPGW-RS-030	
All	SR-OPGW-DI-001	
All	SR-OPGW-DI-010	
All	SR-OPGW-DI-020	PQA requirement.
All	SR-OPGW-QA-001	PQA requirement
All	SR-OPGW-QA-010	Management Requirement
All	SR-OPGW-RM-001	

SW Component Id	SR Id	Comment
N/A	SR-OPGW-FN-540	It is a requirement for the configuration of DAIL / MMOHS.
OPGW - Database	SR-OPGW-FN-001	
OPGW - Database	SR-OPGW-FN-005	
OPGW - Database	SR-OPGW-FN-020	
OPGW - Database	SR-OPGW-IF-070	
OPGW - Database	SR-OPGW-IF-080	
OPGW - Database	SR-OPGW-IF-090	
OPGW - Database	SR-OPGW-PB-001	
OPGW - Database	SR-OPGW-DD-001	
OPGW - HMA Skeleton	SR-OPGW-PB-001	
OPGW - HMA Skeleton [RD-02]	SR-OPGW-FN-500	
OPGW - HMA Skeleton [RD-02]	SR-OPGW-FN-510	
OPGW - HMA Skeleton [RD-02]	SR-OPGW-OP-001	Log files are generated by the Skeleton for each received request. The inspection of log files can be done using an ASCII (or better) an XML editor.
OPGW - HMA Skeleton [RD-02]	SR-OPGW-OP-020	Log files are generated by the Skeleton for each received request. The inspection of log files can be done using an ASCII (or better) an XML editor.
OPGW - HMA Skeleton [RD-02]	SR-OPGW-SP-001	
OPGW - Order Service	SR-OPGW-IF-060	
OPGW - Order Service	SR-OPGW-IF-070	
OPGW - Order Service	SR-OPGW-IF-080	
OPGW - Order Service	SR-OPGW-IF-100	
OPGW - Order Service	SR-OPGW-IF-110	
OPGW - Ordering Service	SR-OPGW-FN-001	
OPGW - Ordering Service	SR-OPGW-FN-005	

SW Component Id	SR Id	Comment
OPGW - Ordering Service	SR-OPGW-FN-010	
OPGW - Ordering Service	SR-OPGW-FN-020	
OPGW - Ordering Service	SR-OPGW-FN-030	
OPGW - Ordering Service	SR-OPGW-PR-001	
OPGW - Ordering Service	SR-OPGW-PR-010	
OPGW - Ordering Service	SR-OPGW-IF-001	
OPGW - Ordering Service	SR-OPGW-IF-010	
OPGW - Ordering Service	SR-OPGW-IF-030	
OPGW - Ordering Service	SR-OPGW-IF-040	
OPGW - Ordering Service	SR-OPGW-IF-050	
OPGW - Ordering Service	SR-OPGW-RM-010	
OPGW - Support Tools	SR-OPGW-FN-001	
OPGW - Support Tools	SR-OPGW-FN-005	
OPGW - Support Tools	SR-OPGW-FN-010	
OPGW - Support Tools	SR-OPGW-FN-020	
OPGW - Support Tools	SR-OPGW-FN-500	
OPGW - Support Tools	SR-OPGW-FN-510	
OPGW - Support Tools	SR-OPGW-FN-520	
OPGW - Support Tools	SR-OPGW-FN-530	
OPGW - Support Tools	SR-OPGW-IF-020	
OPGW - Support Tools	SR-OPGW-IF-040	
OPGW - Support Tools	SR-OPGW-IF-080	
OPGW - Support Tools	SR-OPGW-IF-090	
OPGW - Support Tools	SR-OPGW-SP-001	

Table 6-2: OPGW Software Components vs. OPGW Software Requirements traceability matrix.