ERGO ebRR

Architectural Design Document

Authors:	Jef Vanbockryck, Yaman Ustuntas
Reviewed by:	
Approved by:	

Document Id:	***Issue:***	***Revision:***
ERGOebRR-ADD-4CT	*1.0* *30/12/2008*	*7* *24/04/2009*

Document change record

Issue	Issue date	Pages effected	Reason for change
0.1	30/12/2008	All	Initial draft version
0.2	08/01/2009	All	Formal update of initial draft
0.3	08/01/2009	All	Formal update of initial draft
0.4	22/01/2009	All	Update of architecture with package details
0.5	12/02/2009	All	RIDs
0.6	22/04/2009	All	Update of architecture project structure and deployment packaging; Chapter 5 content added
1.0	24/04/2009	All	Update of version number

Distribution List

Company	Name	Function	N° of copies
Intecs	Simone Gianfranceschi	Project Manager	
ESA	Pier Giorgio Marchetti	ESA Technical Officer	

Table of Content

1. Introduction

1.1. Purpose

This document specifies the technical architecture of the Buddata ebXML Registry-Repository (ebRR) software in the context of the ERGO project.

1.2. Scope

The technical architecture is limited to the Java software architecture of the ebRR. It describes the high level architecture in part 3.1 and all components in detail in part 4.

1.3. Glossary

1.3.1. Abbreviations

Acronym	Extended Form
ebRR	Buddata ebXML Registry-Repository
SOAP	Simple Object Access Protocol (specification from W3C)
HTTP	HyperText Transport Protocol (specification from W3C)
OGC	Open Geospatial Consortium
CSW	Catalogue Web Service (specification from OGC)
JAX-WS	SUN's Java API for XML Web Services
JAX-B	Java Architecture for XML Binding
ebXML	Electronic Business using eXtensible Markup Language (specification from OASIS)
RegRep	Registry-Repository
WS-*	Web Service stack of standards from OASIS
OASIS	Organization for the Advancement of Structured Information Standards
WSS	Web Service Security (specification from OASIS)
XWSS	XML and Web Services Security (name of a SUN project)
SAML	Security Assertion Markup Language (specification from OASIS)
SWA	SOAP with attachments
API	Application Programming Interface
RIM	Registry Information Model
ebRIM	ebXML Registry Information Model (specification from OASIS)
EO	Earth Observations
XML	eXtensible Markup Language (specification from W3C)
GML	Geographic Markup Language (specification from OGC)
WebDav	Web-based Distributed Authoring and Versioning (specification of the IETF)
XSLT	Extensible Stylesheet Language Transformations (specification from W3C)
ISO	International Organization for Standardization
RBAC	Role-Based Access Control
XACML	(specification from OASIS)
SAAJ	SOAP with Attachments API for Java

1.3.2. Definition of Terms

1.4. References

1.4.1. Normative References

[NR1] GMES-DFPR-EOPG_SW-07-0004, Issue 1, Revision 2, Date November 2007 - Implementation of the Cataloguing of ISO Metadata and EO Catalogue service application abRIM profiles based on open source software ERGO, Statement Of Work.

[NR2] ESRIN/Contract n. 21455/08/I-EC, Date 08/02/2008 Implementation of the Cataloguing of ISO Metadata and EO Catalogue service application abRIM profiles based on open source software ERGO.

[NR3] ERGO-MNG-PROP-354-07-SP-PI, issue 1, Revision 0, Date 05/02/2008 ERGO, Management Financial and Administrative Proposal.

[NR4] ERGO-TEC-PROP-354-07-SP-PI, issue 1, Revision 0, Date 05/02/2008 ERGO Technical Proposal

[NR5] ERGO Requirement Baseline, ERG-RB-2100-INT, Issue 1, Revision 1, 06/06/2008

[NR6] OGC Catalogue Services Specification 2.0.0 (with Corrigendum) EO Products Extension Package for ebRim (ISO/TS 15000-3) Profile of CSW 2.0, OGC 06-131, Issue 0, Revision 0.3, 18/08/2006

[NR7] OGC® Cataloguing of ISO Metadata (CIM) Using the ebRIM profile of CS-W, OGC 07-038, Issue 0, Revision 0.7, 10/05/2007

1.4.2. Informative references

[IR1] Software — Part 2: Document requirements definitions (DRDs), ECSS--E--40 Part 2B, 31 March 2005.

[IR2] OASIS ebXML Registry Information Model version 3.0

[IR3] OGC EO Products Extension Package for ebRIM (ISO/TS 15000-3) Profile of CSW 2.0 (0.1.9)

[IR4] OGC CSW-ebRIM Registry Service - Part 1: ebRIM profile of CSW (1.0.0)

[IR5] OGC CSW-ebRIM Registry Service – Part 2: Basic extension package (1.0.0)

[IR6] OGC Catalogue Services Specification (2.0.2)

1.5. Document Overview

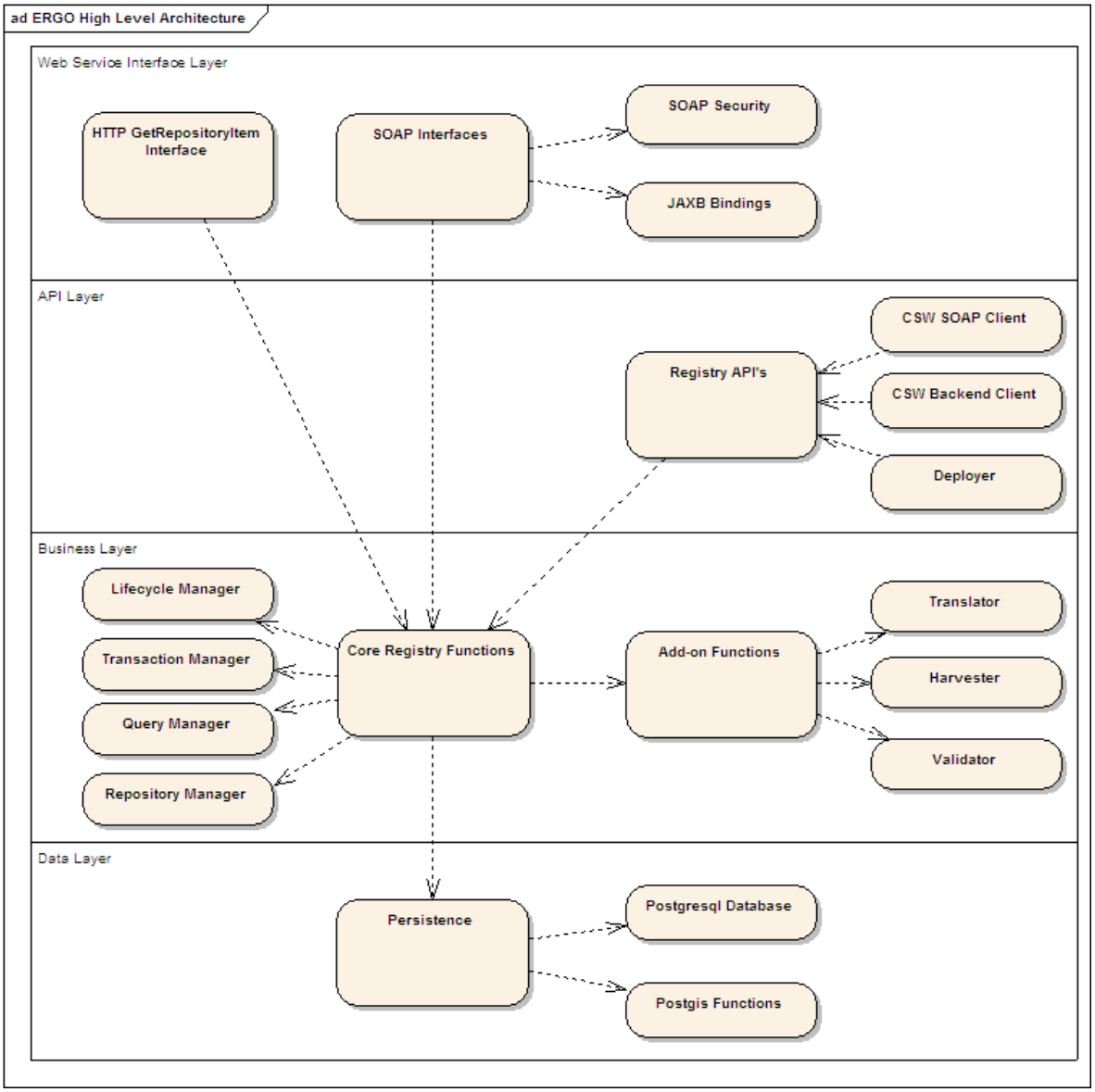
2. SYSTEM DESIGN OVERVIEW

2.1. Software static architecture

The ebRR architecture consists of 4 layers:

- Web Service Interface Layer: SOAP and HTTP-based interfaces for external software to access the ebRR over HTTP
- API layer: Java interface to the core ebRR functions
- Business layer: The basic functionality provided by the ebRR
- Data layer: The data access objects, database and database functions used by the ebRR

2.1.1. High level software architecture view



2.1.1.1. Web Service Interface layer

The Web Service Interface layer presents SOAP interfaces including WS-Security to remote clients and a HTTP interface for getting repository items.

This SOAP interfaces of the ERGO ebRR implement most of the OGC CSW SOAP interface [IR6].

All SOAP Web Services are build on the SUN Metro libraries which support the WS-* OASIS standards. JAX-WS is the most prevalent library of Metro, which provides the basic toolkit for developing Web Services. JAX-B is used for marshalling Java Objects to XML and vice versa, which is used by the Web Services to exchange data and which is specified by the XMLSchemas of OASIS (for the ebXML Registry) and OGC (for the CSW).
XWSS is the SUN Metro security library that supports the following OASIS Web Service Security standards: WSS Basic, WSS UsernameToken, WSS X509, WSS SAML and WSS SWA.

The SOAP interface is directly linked with the ERGO ebRR JAX-B Bindings that represent the various information models taken from the various XMLSchemas for the ebRIM and CSW family of XMLSchemas.

It's important to notice that these JAX-B Bindings use the JAX-B XMLSchema validation capability to validate all incoming XML received via the SOAP Interfaces against the aforementioned XMLSchemas. This allows for validation of the syntax of the submitted XML objects.
Further content validation is performed by the Validator (see "Business Layer").

The HTTP interface for getting repository items also implements the OGC CSW SOAP interface [IR6], but only one operation, the GetRepositoryItem. This is a function not supported by the SOAP binding. It is implemented as a servlet.

2.1.1.2. API layer

The API layer is the Java interface for local Java clients. This can be used by local applications such as the Intecs Toolbox to locally connect to the ERGO ebRR. The API layer contains the Registry API which provides the OGC CSW operations.

The CSW ebRIM Basic Extension Package operations returns metadata content which is 100% compliant with the OGC CSW ebRIM XMLSchema which in turn imports OGC GML XMLSchemas for the geospatial data types (see "Object Model" for more details).
The CSW ebRIM EO Products Extension Package operations returns metadata content which is 100% compliant with the OGC GML Application Schema for EO Products and it's mapping to ebRIM.

The SOAP Web Services use the Registry API as entry point to the ERGO ebRR.

Just like the SOAP Web Services, the Registry API uses the SUN Metro libraries mainly for the JAX-B component, which is responsible for marshalling Java objects to XML and vice versa.
It is possible to run multiple instances of a registry in the ERGO ebRR, for multiple extension packages and for each owner organization.
Configuration of a registry and associated repository is performed by the Deployment API (see further).

The API Layer contains 3 components: CSW Client, CSW Backend Client and the Deployer.

The CSW Client provides a Java client to the SOAP Interface. This client can be used by external applications to integrate the ERGO ebRR in their (Java-based) front-end application.

The CSW Backend Client provides a Java client directly to the Core Functions. This client is used by local applications to integrate the ERGO ebRR directly with the local Java application. In this case, the ERGO ebRR needs a local installation.

The Deployer is a simple Java client to initiate a deployment of a new instance of a registry/repository.

2.1.1.3. Business layer

The Business layer contains the Core Registry functions (which are minimally required to comply with the ebXML RegRep specification) and the Registry Add-ons which are required for the ERGO project and geospatial applications in general. The Registry API uses all Core Registry functions.

The *Core Registry Functions* have the following 4 components: the Lifecycle Manager, the Query Manager, the Transaction Manager and the Repository Manager.

The Lifecycle Manager is the main controller of the ERGO ebRR and manages the state and versioning of a registry object.

The Query Manager contains the functions for generating queries and querying ebXML Registry objects via Persistence (see "Persistence"). It is to be noted that this Query Manager is extended with query functions as defined by the ERGO user requirements. These functions are NOT part of the ebXML RegRep specification but are needed to support the OGC CSW interface.

The Transaction Manager contains functions to insert, update and delete registry objects and support both the ebXML RS and the OGC CSW transaction functionality.

The Repository Manager contains functions to store and retrieve repository items. Such items are e.g. the originally harvested external non-ebRIM metadata files (see “Harvester”).

The *Registry Add-ons* contain the following 3 components: the Translator, the Harvester and the Validator.

The Harvester is responsible for fetching external metadata XML files and digesting them into the ERGO ebRR. This Harvester is capable of harvesting non-ebRIM metadata which complies to the ISO19135 XML specification and the GML EO Application XML specification. Offcourse, the Harvester is also capable of harvesting valid ebRIM metadata from other ebXML If the Harvester detects existing metadata (based on specific metadata identifiers) it will "update" this metadata. This update will in fact be a "delete (off old metadata) and insert (of new metadata)".

The Harvester is closely related with the Translator, which translates non-ebRIM objects into ebRIM objects. Thus the Harvester calls the Translator first before ingesting the (translated) ebRIM objects in the ERGO ebRR.

The Validator is responsible for validating ebRIM extrinsic objects as defined by extension packages. This validation goes further then the XMLSchema validation which is not capable of validation actual content (only structure defined in the XMLSchema). The Validator validates slots, slot data types, association types and associated objects (check if they exist).

The *Security Functions* are extending the basic WS-S security and SAML handling - which are provided by the SUN Metro libraries – to allow for calling out external (non ERGO ebRR) Security Services acting as Policy Enforcement Points (PEP's). Such PEP return either a true or false to the ERGO ebRR Security Handler, indicating whether a client has or has no access to the requested Web Service Operation. These security functions use the WS-Policy standard to configure security settings.

2.1.1.4. Data layer

The Data layer contains a Java part containing logic for managing data access objects (Persistence) and a part on the PostgreSQL database, which is responsible for storing the relational data.

Persistence is based on internal ebRR Data Access Objects. These are responsible for the object-relational mapping between Java objects and the relational data from the PostgreSQL database. The Persistence function is responsible for execution of this activity.

Postgis functions - that support geospatial data types - and additional stored procedures are used to support the complex queries required by the ERGO specification. These are directly used by the Query Manager.

The PostgreSQL database is used to store and manage the relational data which is mapped to Java objects by the ebRR Data Access Objects and marshalled to ebRIM XML via JAX-B. It is to be noted that there's a particular binding with this database because the usage of the special geospatial PostgreSQL functions and stored procedures.

2.1.2. Static system architecture

The static system architecture consists of 4 software packages: Web Application Package, CSW SOAP Client Package, CSW Backend Client Package and the Postgressql Database Server.

The software top-level architecture design (Chapter 3) will go in detail on the content of the3 first packages which are specific to ebRR.
The PostgreSQL Database Server is an existing COTS and is not further detailed.

The next section shows how these packages can be installed in 3 different type of deployments.

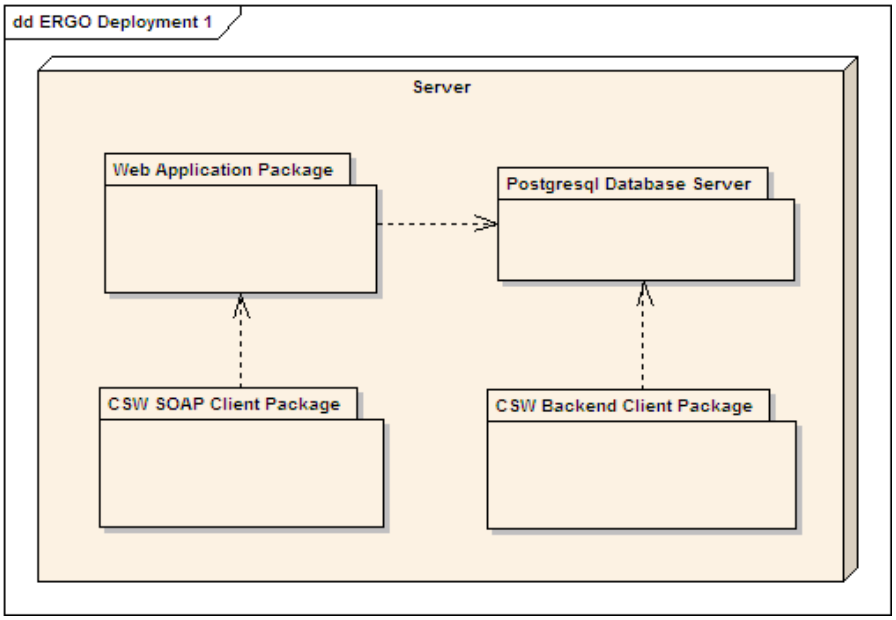
2.1.3. Deployment options

The software can be deployed on either the same physical machine (single machine set-up), on two machines (dual machine set-up) or three machines (triple machine set-up).
All software runs on both Windows and Linux platforms, from which only the PostgreSQL installation differs.

The preferred platform is Linux for which an installer will be provided.

2.1.3.1. Single machine set-up

In this setup alls software is installed on one physical machine.



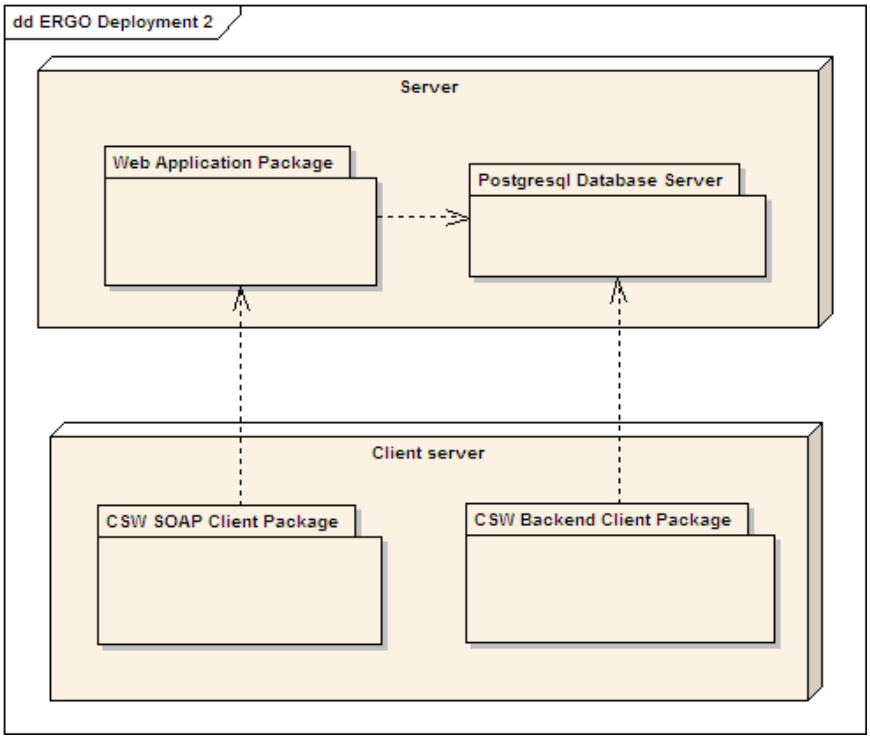
These are the communication requirements for this set-up:

- Connection requirements: Via the clients
- List of ports used:
  - For Java application: 8080 (Tomcat default)
  - For Postgres database: 5432 (Postgresql default)

- Firewall constraints:
  - Tomcat port requires outside access.

2.1.3.2. Dual machine set-up

In this set-up, the Web Application Package and Postgresql Database Server are installed on one physical machine (server).
The clients are installed on a second machine (client server).

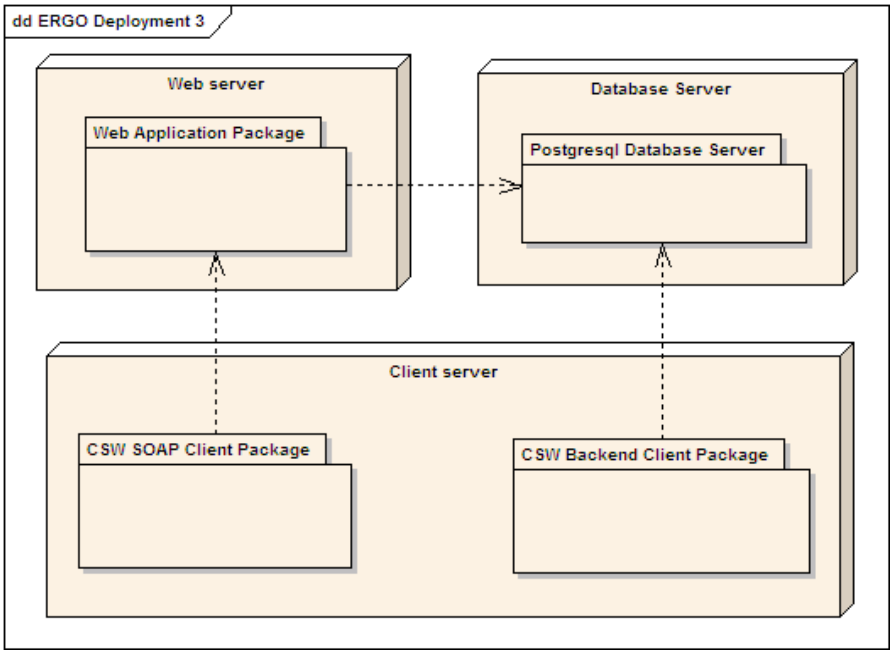


These are the communication requirements for this set-up:

- Connection requirements: Via the clients
- List of ports used:
  - For Java application: 8080 (Tomcat default)
  - For Postgres database: 5432 (Postgresql default)
- Firewall constraints:
  - Tomcat port requires outside access.
  - In case the DSW Backend Client is used, there's a requirement to access the Postgresql port from the client device.

2.1.3.3. Tripple machine set-up

In this set-up, the Web Application Package is installed on one machine (Web Server), the Postgressql Database Server on a second machine (Database server) and the clients on a third machine (Client server).



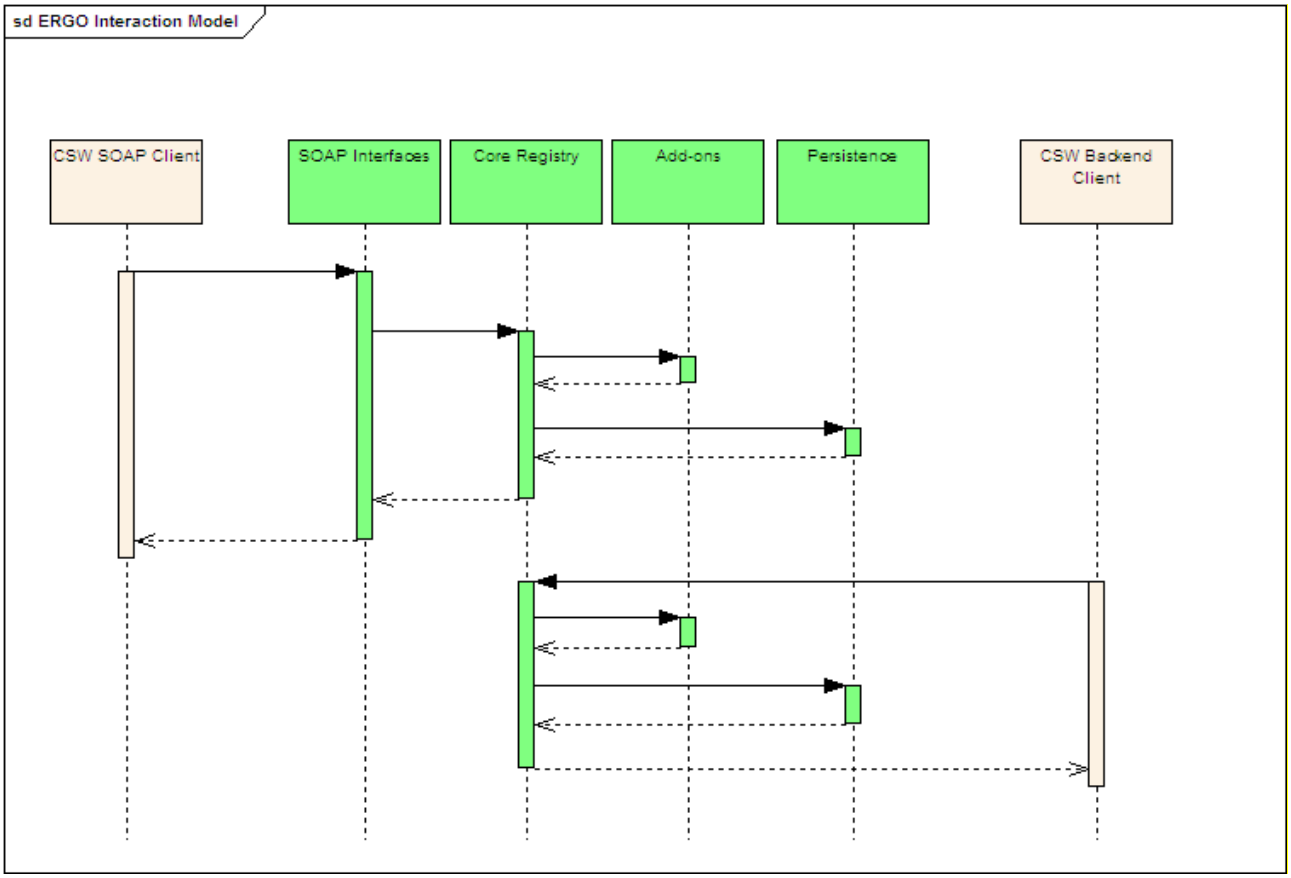
These are the communication requirements for this set-up:

- Connection requirements: Via the clients
- List of ports used:
  - For Java application: 8080 (Tomcat default)
  - For Postgres database: 5432 (Postgresql default)
- Firewall constraints:
  - Tomcat port requires outside access
  - Tomcat requires access to Postgresql port.
  - In case the DSW Backend Client is used, there's a requirement to access the Postgresql port from the client device.

2.2. Software dynamic architecture

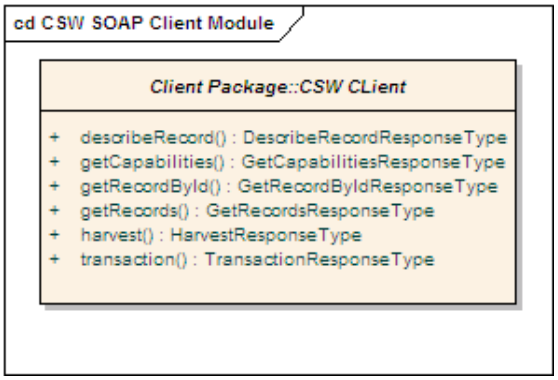
The diagram below shows the interaction between the main components of the ERGO ebRR.

The clients interact with the Core Registry in the same way, but they access the core in a different way. The CSW SOAP Client uses the SOAP Interfaces to communicate with the Core Registry. The CSW Backend Client, however, communicates directly with the Core Registry.
In both cases, the Core Registry communicates wit the Add-on functions and the Persistence functions.



2.3. Interfaces context

All interfaces, either using SOAP or connecting directly to the Core Registry, are based on the same OGC CSW interface and use both the CSW Client module (as showed in diagram below).



The external interfaces to the ERGO ebRR are specified by [IR6] using SOAP encoding. The SOAP Interfaces of the ERGO ebRR provide this external interface.
On top of these ERGO ebRR SOAP Interfaces is a CSW SOAP Client which can be used by any Java application to connect via the SOAP Interfaces to the ERGO ebRR.

There is also an internal interface to the ERGO ebRR provided by the CSW Backend Client. This Java interface connects directly to the Core Registry functions of the ERGO ebRR and provides an interface to local Java applications.

2.4. Memory and CPU budget

The following minimal server requirements need to be met in terms of memery and CPU budget:

- Single machine configuration:
  - Linux server.
  - 2 GB RAM (1GB reserved for Postgresql, 512MB reserved for Tomcat)
  - Dual-core CPU.
  - Minimum 5 GB disk space
- Dual machine configuration:
  - For Java application server:
    - Linux server
    - 1GB RAM
    - Dual-core CPU
    - Minimum 2GB of disk space
  - For Postgresql database server:
    - Linux server
    - 1GB RAM
    - Dual-core CPU
    - Minimum 3GB of disk space

2.5. Design standards, conventions and procedures

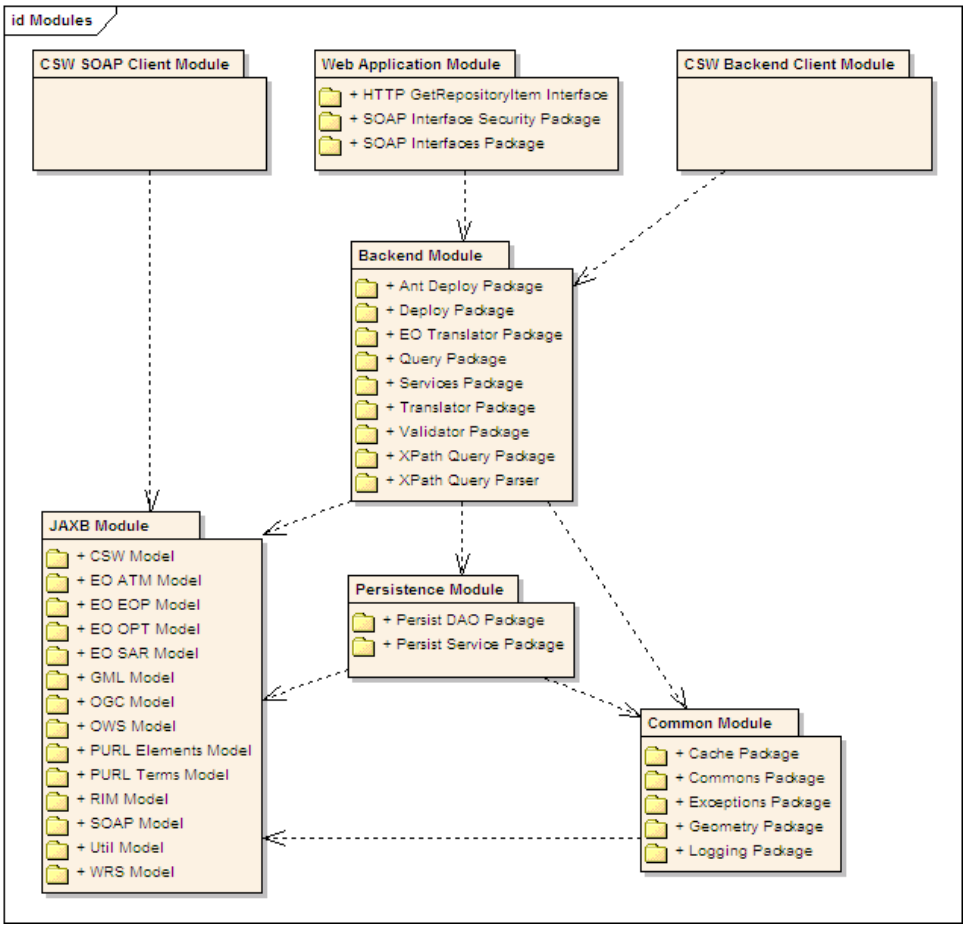
The UML2.0 standard is used in UML drawing tools to outline software architecture, including the component structure and deployment options.

3. SOFTWARE TOP-LEVEL ARCHITECTURAL DESIGN

3.1. Packages overview

The ERGO ebRR software architecture contains 7 Modules as shown in the next diagram:

- CSW SOAP Client Module
- CSW Backend Client Module
- Web Application Module
- Backend Module
- JAXB Module
- Common Module
- Persistence Module

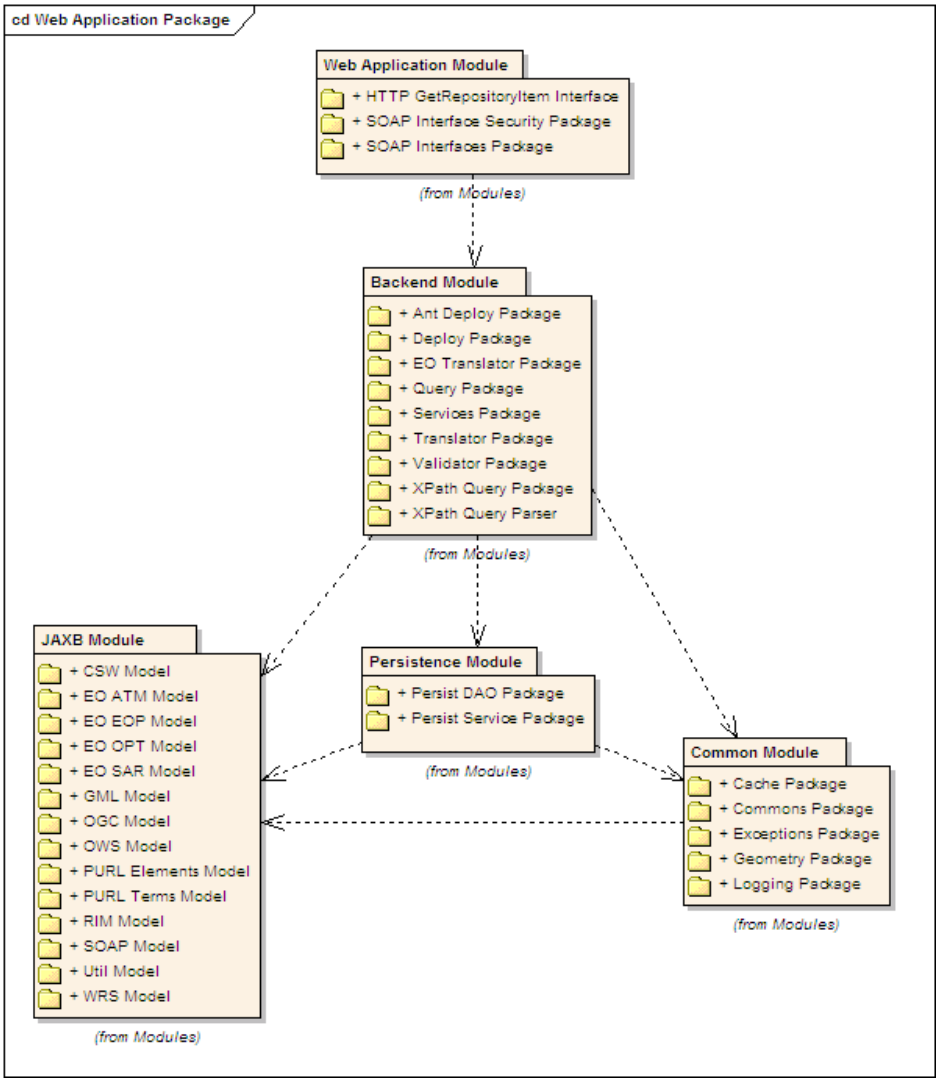


The dependencies between the Modules are shown in the diagram by the dotted arrows. It's important to notice the direction of these dependencies. A Module depends on another Module in the direction of the arrow.

These modules are further "packaged" into 4 packages that are used by the various deployment options (see 2.1.3):

- Web Application Package
- Postgressql Database Server
- CSW SOAP Client Package
- CSW Backend Client Package

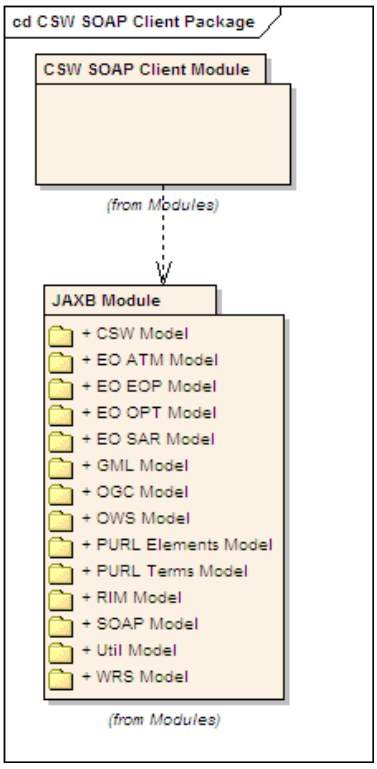
3.1.1. Web Application Package



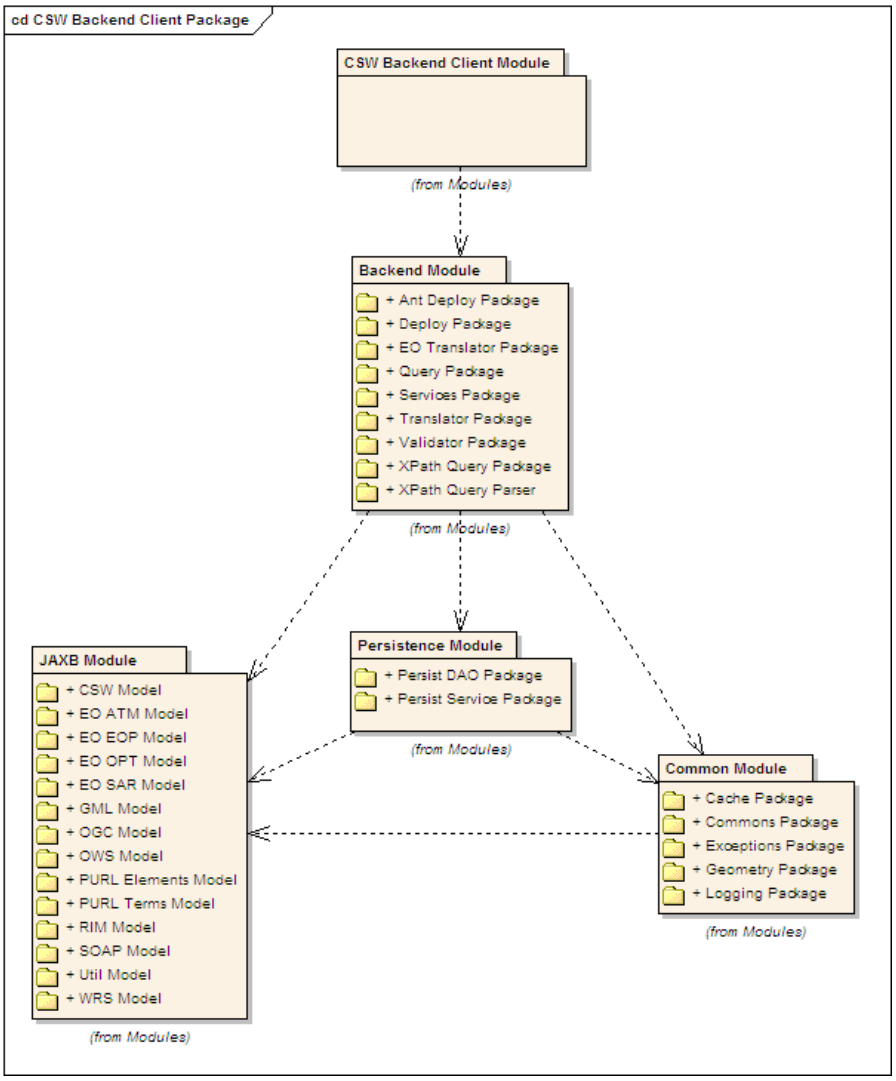
3.1.2. Postgress Database Server

This is not further specified, because it is an external COTS.

3.1.3. CSW SOAP Client Package



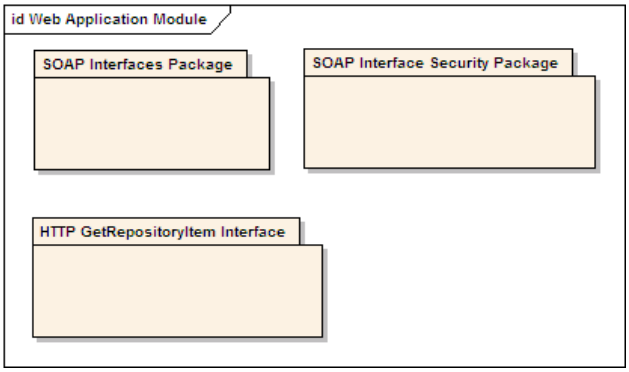
3.1.4. CSW Backend Client Package



3.2. *Modules overview*

3.2.1. Web Application Module

The Web Application module contains 3 software packages as shown in the diagram below:

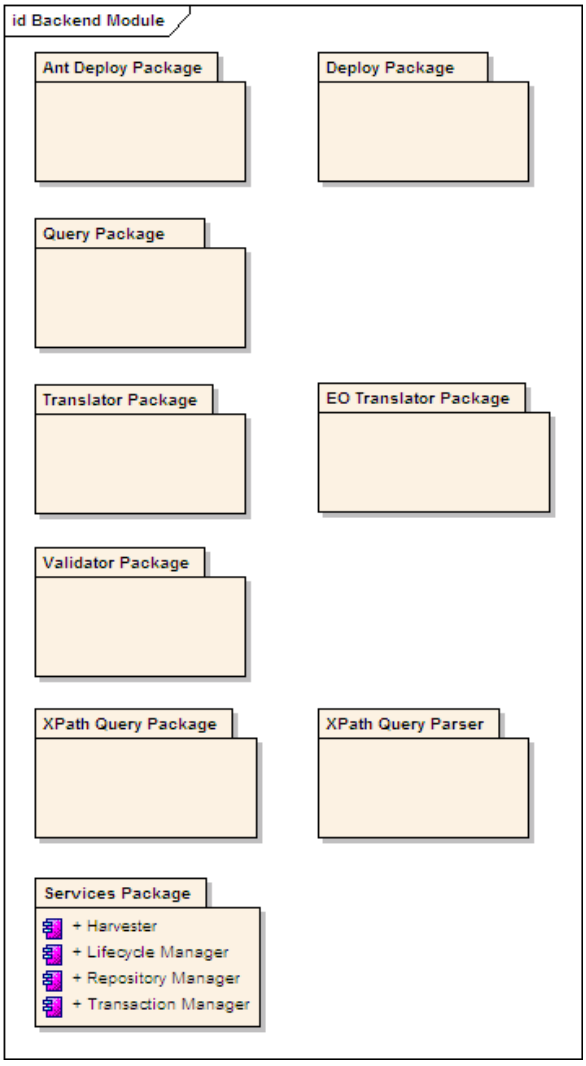


These packages are

- SOAP Interface Package
- SOAP Interface Security Package
- HTTP GetRepositoryItem Interface

3.2.2. Backend Module

The Backend Module contains the Core Registry and Add-on functions. They are organized into seperated packages, as shown in the diagram below:



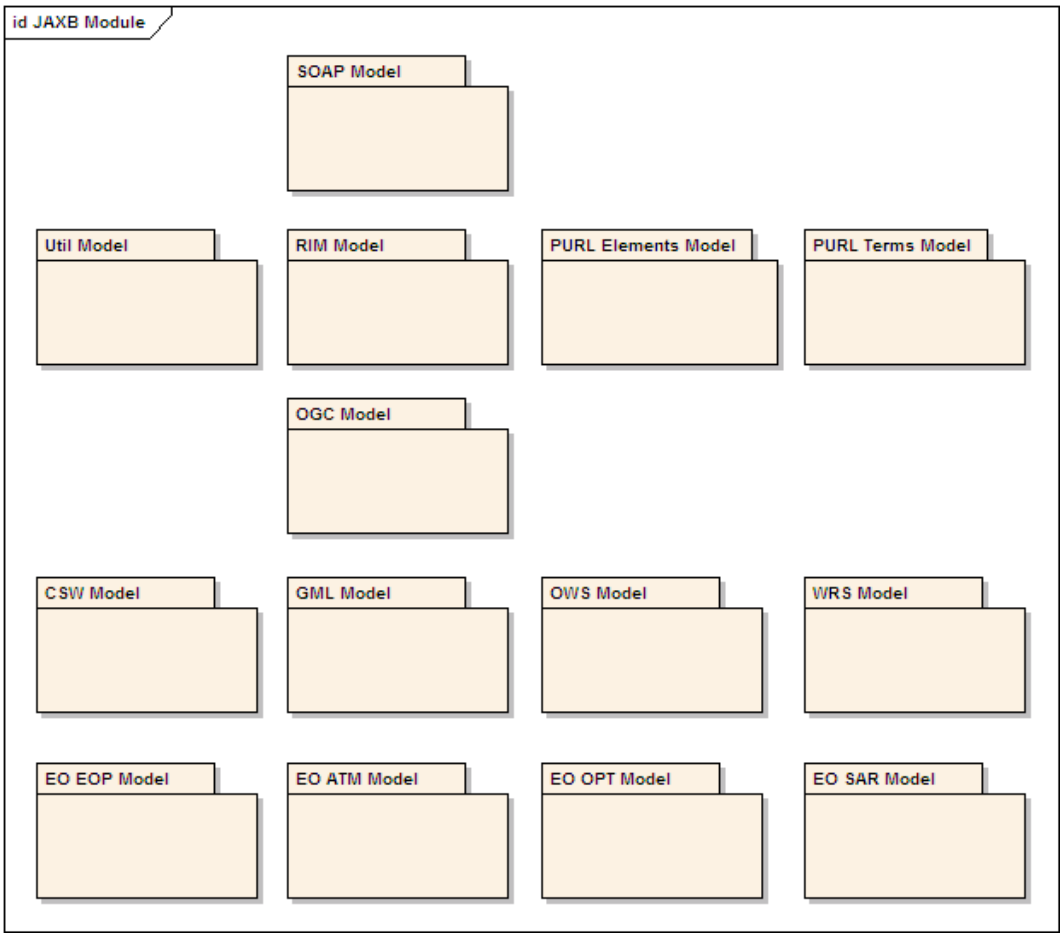
The software packages inside the Backend Module are:

- Ant Deploy Package
- Deploy Package
- Query Package
- Translator Package
- EO Translator Package
- Validator Package

- XPath Query Package
- XPathy Query Parser
- Services Package, with main classes: Harvester, Lifecycle Manager, Repository Manager and Transaction Manager.

3.2.3. JAXB Module

This module contains all the software packages that represent the information model XMLSchema bindings as Java classes.



These models support the OGC CSW interface and extension packages Basic and EO. If later other extension packages are supported (e.g. CIM), then this JAXB Package will be extended with additional models and bindings.
Currently this is the list of supported JAXB models (as shown in the diagram above):

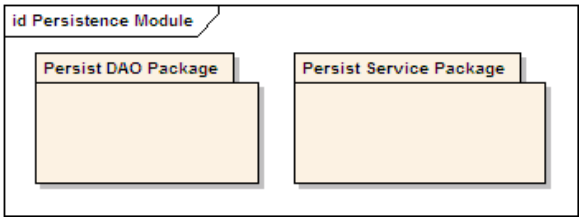
- SOAP Model
- Util Model
- RIM Model
- PURL Elements Model
- PURL Terms Model
- OGC Model
- CSW Model

- GML Model
- OWS Model
- WRS Model
- EO ATM Model
- EO EOP Model
- EO OPT Model
- EO SAR Model

3.2.4. Persistence Module

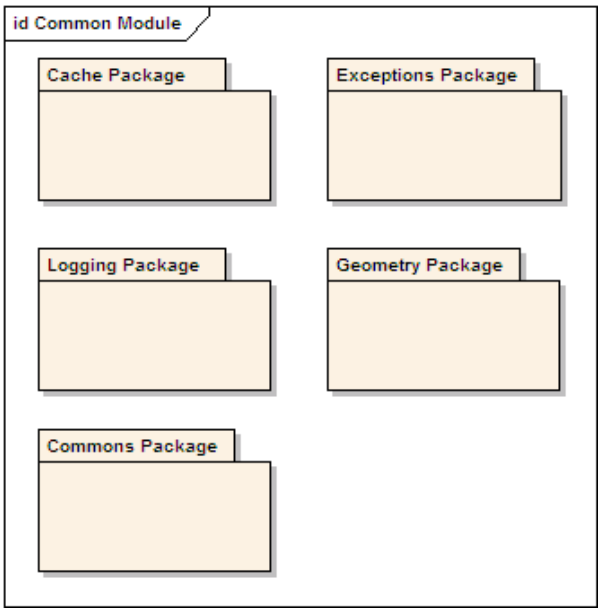
This module contains only two software packages dealing with data persistency:

- Persist DAO Package
- Persist Service Package



3.2.5. Common Module

The Common module contains a number of software packages which are tools and utilities that are commonly used by most other packages and their classes.



Currently 5 packages are inside the Common Module (as shown in the diagram above):

- Cache Package
- Exceptions Package
- Logging Package
- Geometry Package

- Commons Package

4. Software architectural design

4.1. Software Item components

1.1. Software Item components

1.1.1. SOAP Interface

1.1.1.1. Type

Containing Java classes from the Web Application Module.

1.1.1.2. Purpose

Java implementation of the CSW SOAP interface.

1.1.2. SOAP Interface Security

1.1.2.1. Type

Containing Java classes from the Web Application Module.

1.1.2.2. Purpose

WSS implementation ont top of the CSW SOAP interface.

1.1.3. HTTP Interface

1.1.3.1. Type

Containing Java classes from the Web Application Module.

1.1.3.2. Purpose

Java implementation of the CSW GetRepositoryItem interface via HTTP as a servlet.

1.1.4. Services

1.1.4.1. Type

Containing Java classes from the Backend Module.

1.1.4.2. Purpose

Contains service implementations such as:

- Lifeceycle Manager: Manages the life cycle of registry objects
- Harvester: Harvests input metadata in different formats to the registry in RIM format
- Transaction Manager: Fetches input metadata from a third party source to store it in the registry

- Repository Manager: Stores external metadata objects (which were collected via the Harvester)

1.1.5. Query

1.1.5.1. Type

Containing Java classes from the Backend Module.

1.1.5.2. Purpose

High level package with the purpose to translate OGC Filter queries to plain SQL queries.

1.1.6. XPath Query

1.1.6.1. Type

Containing Java classes from the Backend Module.

1.1.6.2. Purpose

Helps to translate Xpaths defined in OGC Filter queries to SQL.

1.1.7. XPath Query Parser

1.1.7.1. Type

Containing Java classes from the Backend Module.

1.1.7.2. Purpose

Parses Xpaths in OGC Filter queries to be processed to a SQL query.

1.1.8. Translator

1.1.8.1. Type

Containing Java classes from the Backend Module.

1.1.8.2. Purpose

Package containing translators from one XML format to RIM model.

1.1.9. EO Translator

1.1.9.1. Type

Containing Java classes from the Backend Module.

1.1.9.2. Purpose

Translates EO JAXB bindings to the RIM model.

1.1.10. Validator

1.1.10.1. Type

Containing Java classes from the Backend Module.

1.1.10.2. Purpose

Validates the RIM model to be inserted or updated. Checks references and associations for valid content.

1.1.11. Deploy

1.1.11.1. Type

Containing Java classes from the Backend Module.

1.1.11.2. Purpose

Allows to deploy another instance of the registry (database and web application):

- Configure application
- Create database from Postgis template in local or remote database
- Create tables in database in local or remote database
- Compile and build all packages
- Deploy to local or remote Tomcat
- Insert registry initialization data (ClassificationScheme/Node, EO RIM model, ...)

1.1.12. Ant Deploy

1.1.12.1. Type

Containing Java classes from the Backend Module.

1.1.12.2. Purpose

Custom Ant tasks help in the deployment of the application.

1.1.13. SOAP Model

1.1.13.1. Type

Containing Java classes from the JAXB Module.

1.1.13.2. Purpose

Contains the Java interfaces for the CSW interface.

1.1.14. Util Model

1.1.14.1. Type

Containing Java classes from the JAXB Module.

1.1.14.2. Purpose

Utility classes to ease the use of JAXB bindings.

1.1.15. RIM Model

1.1.15.1. Type

Containing Java classes from the JAXB Module.

1.1.15.2. Purpose

JAXB bindings for the RIM model.

1.1.16. PURL Elements Model

1.1.16.1. Type

Containing Java classes from the JAXB Module.

1.1.16.2. Purpose

JAXB bindings for the PURL model.

1.1.17. PURL Terms Model

1.1.17.1. Type

Containing Java classes from the JAXB Module.

1.1.17.2. Purpose

JAXB bindings for the PURL Terms.

1.1.18. OGC Model

1.1.18.1. Type

Containing Java classes from the JAXB Module.

1.1.18.2. Purpose

JAXB binding for the OGC namespace.

1.1.19. CSW Model

1.1.19.1. Type

Containing Java classes from the JAXB Module.

1.1.19.2. Purpose

JAXB bindings for the CSW model.

1.1.20. GML Model

1.1.20.1. Type

Containing Java classes from the JAXB Module.

1.1.20.2. Purpose

JAXB bindings for the GML model.

1.1.21. OWS Model

1.1.21.1. Type

Containing Java classes from the JAXB Module.

1.1.21.2. Purpose

JAXB bindings for the OWS model.

1.1.22. EO ATM Model

1.1.22.1. Type

Containing Java classes from the JAXB Module.

1.1.22.2. Purpose

JAXB bindings for the EO ATM model.

1.1.23. EO EOP Model

1.1.23.1. Type

Containing Java classes from the JAXB Module.

1.1.23.2. Purpose

JAXB bindings for the EO EOP model.

1.1.24. EO OPT Model

1.1.24.1. Type

Containing Java classes from the JAXB Module.

1.1.24.2. Purpose

JAXB bindings for the EO OPT model.

1.1.25. EO SAR Model

1.1.25.1. Type

Containing Java classes from the JAXB Module.

1.1.25.2. Purpose

JAXB bindings for the EO SAR model.

1.1.26. Persist DAO

1.1.26.1. Type

Containing Java classes from the Persistence Module.

1.1.26.2. Purpose

Contains the Data Access Objects which are used for exchanging data with the database.

1.1.27. Persist Service

1.1.27.1. Type

Containing Java classes from the Persistence Module.

1.1.27.2. Purpose

Contains the actions to connect and interact with the database (insert, update, delete).

1.1.28. Cache

1.1.28.1. Type

Containing Java classes from the Common Module.

1.1.28.2. Purpose

Helper classes to handle caching.

1.1.29. Exceptions

1.1.29.1. Type

Containing Java classes from the Common Module.

1.1.29.2. Purpose

General exceptions used by the registry.

1.1.30. Logging

1.1.30.1. Type

Containing Java classes from the Common Module.

1.1.30.2. Purpose

Utility classes to set up loggings.

1.1.31. Geometry

1.1.31.1. Type

Containing Java classes from the Common Module.

1.1.31.2. Purpose

Geometry utility classes to help transform GML geometry to Postgis geometry and visa versa.

1.1.32. Commons

1.1.32.1. Type

Containing Java classes from the Common Module.

1.1.32.2. Purpose

Utility classes used by various other classes.

4.2. Internal Interfaces identification

This part is intentionally left blank.

4.2.1.1. Internal Interface specification

This part is intentionally left blank.

5. Software Requirements Traceability Matrix

This chapter contains a table in which the Software Requirements are mapped against the software components of this architecture as outlined in chapter 4 of this document in order to provide a Software Requirements Traceability Matrix.

Software requirements	Software components
ERG-SR-EBR-FUN-010	All
ERG-SR-EBR-FUN-020	1.1.15 up to 1.1.25
ERG-SR-EBR-FUN-030	1.1.4
ERG-SR-EBR-FUN-040	1.1.4
ERG-SR-EBR-FUN-050	1.1.5
ERG-SR-EBR-FUN-060	1.1.4
ERG-SR-EBR-FUN-070	1.1.5
ERG-SR-EBR-PER-060	N/A
ERG-SR-EBR-PER-061	1.1.5
ERG-SR-EBR-PER-062	1.1.5
ERG-SR-EBR-PER-063	1.1.5
ERG-SR-EBR-PER-064	1.1.5
ERG-SR-EBR-DES-250	Applies to all
ERG-SR-EBR-DES-260	Applies to all
ERG-SR-EBR-INT-070	1.1.15 up to 1.1.25
ERG-SR-EBR-INT-080	1.1.15
ERG-SR-EBR-INT-090	?
ERG-SR-EBR-INT-100	1.1.1
ERG-SR-EBR-INT-110	1.1.1
ERG-SR-EBR-OPE-120	1.1.11
ERG-SR-EBR-OPE-130	1.1.11
ERG-SR-EBR-OPE-140	1.1.4, 1.1.5
ERG-SR-EBR-OPE-150	1.1.5
ERG-SR-EBR-OPE-160	1.1.5, 1.1.6, 1.1.7
ERG-SR-EBR-OPE-170	1.1.5
ERG-SR-EBR-OPE-180	1.1.5
ERG-SR-EBR-OPE-190	1.1.5
ERG-SR-EBR-OPE-200	1.1.5
ERG-SR-EBR-OPE-210	?
ERG-SR-EBR-OPE-220	1.1.10
ERG-SR-EBR-OPE-230	1.1.5
ERG-SR-EBR-OPE-240	1.1.2
ERG-SR-EBR-OPE-250	?
ERG-SR-EBR-SEC-270	1.1.2
ERG-SR-EBR-SEC-280	Applies to all
ERG-SR-EBR-SEC-290	Applies to all
ERG-SR-EBR-SEC-300	Applies to all

ERG-SR-EBR-DDB-310	N/A
ERG-SR-EBR-ADP-320	N/A
ERG-SR-EBR-ADP-330	N/A
ERG-SR-EBR-OTH-340	N/A
ERG-SR-EBR-VAL-350	N/A