Collaborative Large-scale Integrating Project

Open Platform for EvolutioNary Certification Of Safety-critical Systems

Implementation of the evidence management service infrastructure

D6.6

Work Package:	WP6: Evolutionary Evidential Chain
Dissemination level:	Public
Status:	Final
Date:	March 13th, 2015
Responsible partner:	Janusz Studzizba (Parasoft S.A.)
Contact information:	januszst@parasoft.com

Contributors

Names	Organisation
Dariusz Oszczędłowski, Janusz Studziżba	Parasoft S.A.
Jose Luis de la Vara, Sunil Nair	Simula Research Laboratory
Jan Mauersberger	IKV++
Angel López, Huáscar Espinoza, Xabier Larrucea	Tecnalia
Jérôme Lambourg	Adacore

Document History

Version	Date	Remarks
V0.1	2014-05-27	Document creation, initial ToC, and initial content
V0.2	2014-05-30	Initial text for each chapter provided
V0.3	2014-06-13	Implementation architecture, Impact Analysis, and Development manual described
V0.4	2014-06-17	Simula contribution pasted. Technologies used in server implementation described.
V0.5	2014-06-30	Tecnalia contribution added. Technologies used in client implementation described. Review by Huáscar.
V0.6	2014-07-08	Adacore contribution merged into the document.
V0.7	2014-07-24	Review feedback applied
V1.0	2014-08-25	Deliverable finalisation after PB review. Xabier
V1.2	2014-12-10	Updated with 3rd prototype implementation information.
V1.4	2015-03-13	Updated to align with the final status of work

TABLE OF CONTENTS

List of Figures

Abbreviations

API	Application programming interface
CCL	Common Certification Language
DAO	Data Access Object
DX.Y	OPENCOSS deliverable X.Y
DoW	Description of Work
EMF	Eclipse Modelling Framework
GUI	Graphical User Interface
HTTP	Hypertext Transfer Protocol
OSLC	Open Services for Lifecycle Collaboration
QM	The Qualifying Machine
REST	Representational State Transfer
SACM	Structured Assurance Case Metamodel
SVN	Subversion
TX.Y	OPENCOSS task X.Y
V&V	Verification and Validation
WP	OPENCOSS Work Package
EMF	Eclipse Modeling Framework
EEF	Extended Editing Framework
CDO	Connected Data Objects

Executive Summary

This document presents the sixth deliverable of WP6, which aims to specify and implement a management infrastructure for safety certification evidential chain. The deliverable is a continuation and broad extension of D6.5, which showed the intermediate achievements of the evidence management implementation effort. The goal of this document is to present and summarize results of the implementation of the evidence management functionality in OPENCOSS tool platform.

As explained in D6.5 document, OPENCOSS platform software development has been divided into 3 phases, each resulting in a functional and ready-to-use tool prototype. The current results from T6.4 correspond to the 3rd prototype of the OPENCOSS tool platform. This document presents in detail the functionalities implemented in the 3rd prototype for evidence management, its software architecture, the technology used, and source code references.

Other important parts of D6.6 document are:

- User Manual, which describes how to install and use the OPENCOSS tool platform.
- Developer Manual, which describes how to set up the software development environment of the tool platform. This manual is split into one document for the client part and another for the server part.

1 Functionality implemented in OPENCOSS Platform tools

The following pieces of functionality have been implemented as the result of software development done for the OPENCOSS platform 3rd prototype:

- Client-server infrastructure
- Central data storage used by both server and clients
- Evidence Gap Analysis web report
- Evidence items Change Impact Analysis
- Events mechanism framework in the server
- REST API serving evidence information
- EEF Editor (Forms Editor) for evidence management
- Integration of the evidence editor with the Impact Analysis functionality

The software implementation of the above functionality is described in detail in the following chapters.

With regards to architectural components, Figure 1 (based on a similar one in D6.3) shows the components that have been implemented in 1st, 2nd and 3rd prototypes. The implementation of the components presented in green has been done in the 1st prototype and is described in D6.5 document. A description of the pieces of software developed for the components presented in blue (i.e., implemented in the 2nd and 3rd prototype) is provided below.

EvidenceReporter component provides reporting functionalities for OPENCOSS tool users. This module has been materialised in the Gap Analysis web report. The report presents evidence properties and evidence evaluations. The detailed description of the report implementation is presented in the OPENCOSS Platform tool user manual chapter. The software development technology used for the web report implementation is described in Web framework.

EvidenceAnalyser component provides evidence-specific functionality for Change Impact Analysis. The implementation of Change Impact Analysis is described in Change Impact Analysis chapter. This component engine is used by client editors in order to visualize how evidence modification affects the related artefacts.

TraceLinkEngine component is in charge of storing dependency information as traceability links. It has been implemented together with Impact Analyser engine. The software developed is described in Change Impact Analysis chapter.

EvidenceComplianceManager supports the measurement of the level of compliance of artefacts with standards, rules, regulations, or company-specific practices. A prototype of this module is implemented in Gap Analysis report, which is described in OPENCOSS Platform tool user manual. As an example, Figure 2 shows Gap Analysis web report user interface for presenting project safety artefacts compliance to a safety standard.

GUIEvidenceAnalyser component supports visualization, configuration, and execution of change impact analysis and compliance check. In OPENCOSS 2nd and 3rd prototype tool, it has been implemented as Gap Analysis report in web interface and Impact Analysis runner in client editors.

DataChangeManager module has been materialized in a database storage and CDO module serving CCL modelled data persistence. The CDO module supports data versioning of each item

stored. For the 3rd prototype, the data versioning is done in the storage, so all the data versions are preserved, however they are not presented in GUI modules yet. Only the last version of a given entity is presented. CDO provides an API that can be used in future implementations to visualize the versions of each piece of data saved in the storage. See Client-server architecture with central data storage for further descriptions of the data persistence in OPENCOSS platform.

GUI_EvidenceEditor component provides the user interface to create, modify and delete evidence data to the CDO repository using EMF/EEF/CDO technology. This editor is integrated with a SVN evidence repository and it connects the artefacts model to the actual artefact objects (word documents, excel files, pdfs, log files and so on), which are stored in the SNV evidence server. This editor is also integrated with the Impact Analysis module to inform the user about any evidence modification that may affect the related artefacts.

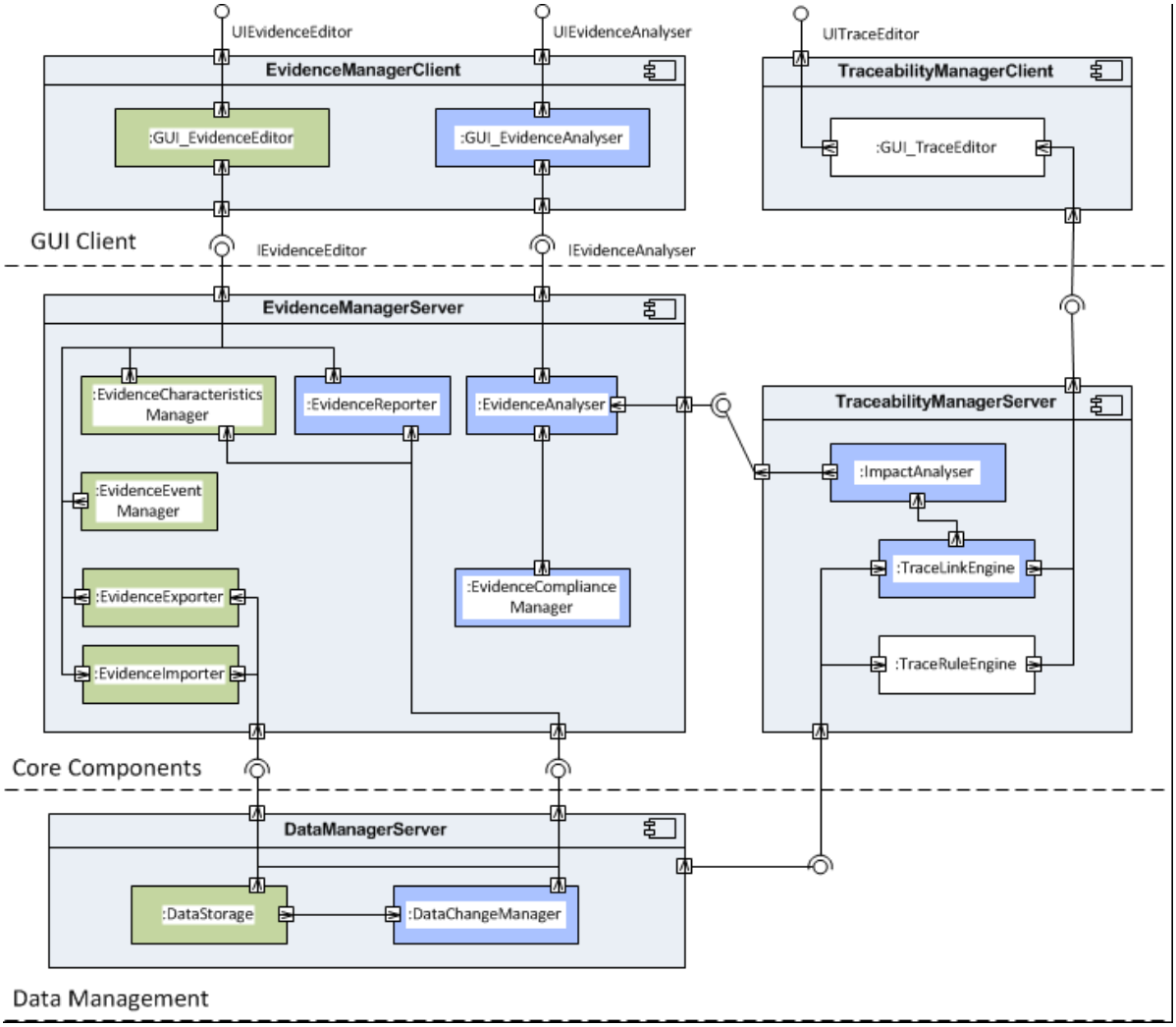


Figure 1. OPENCOSS Tool Components - components implemented in 1st prototype are presented in green while 2nd + 3rd prototype in blue

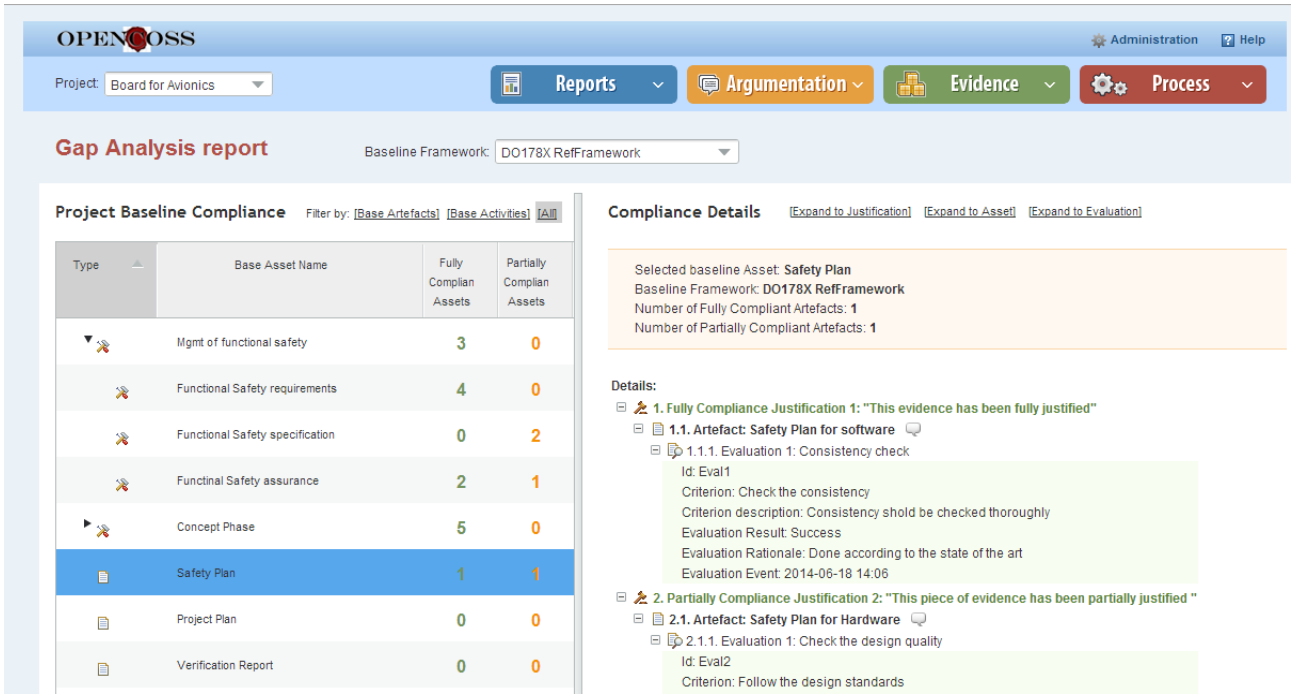


Figure 2. Gap Analysis web report GUI

2 Implementation architecture and source code description

The documents of the D2.3, D6.3, and D6.4 deliverables presented the OPENCOSS platform tool architecture. The designed structure has been implemented in the scope of T6.4 and the developed software is presented in this document. This chapter presents the detailed implementation architecture, and the software development technologies that have been used, their configuration, settings, and source code references.

Note: D6.4 contains a design for OPENCOSS tool integration with external evidence tools. Initial implementation effort has been performed in this direction. In the 1st prototype integration with Subversion (SVN) as an example evidence storage has been implemented. In the 2nd prototype initial integration with QM has been addressed. This is described in Evidence REST API and initial integration with QM. Further implementation in this area was made for the 3rd prototype.

2.1 Client-server architecture with central data storage

The diagram in the next page depicts the deployment and communication between the main implementation modules of the OPENCOSS Platform. Each module technology as well as communication protocols are presented below.

Data persistence in OPENCOSS platform tools is provided by a CDO repository. CDO is a Java model persistence solution for EMF models and metamodels. CDO model repository documentation can be found at http://www.eclipse.org/cdo/documentation/.

CDO can use various relational databases as its backend storage. In a default OPENCOSS platform installation, PostgreSQL database is used. It is accessed by CDO repository using JDBC protocol at 5432 port on the database side by default.

There are two general types of OPENCOSS platform modules which access the CDO repository: OPENCOSS clients (used on client machines) and OPENCOSS server (installed on the server machine).

The CDO server is accessed by its clients using CDO protocol. By default, the 2036 port on the CDO server side is used. The implementation of the CDO Server is described in the CDO Server implementation chapter.

OPENCOSS platform client tools have been implemented as Eclipse plugins and are supposed to be installed on user machines. They have been implemented using EMF (Eclipse Modelling Framework) technology and auxiliary technologies like EEF (Extended Editing Framework) and GMF (Graphical Modelling Framework). Development documentation for these technologies can be found on the following sites, respectively:

- http://www.eclipse.org/modeling/emf/
- http://www.eclipse.org/modeling/emft/?project=eef
- http://www.eclipse.org/modeling/gmp/

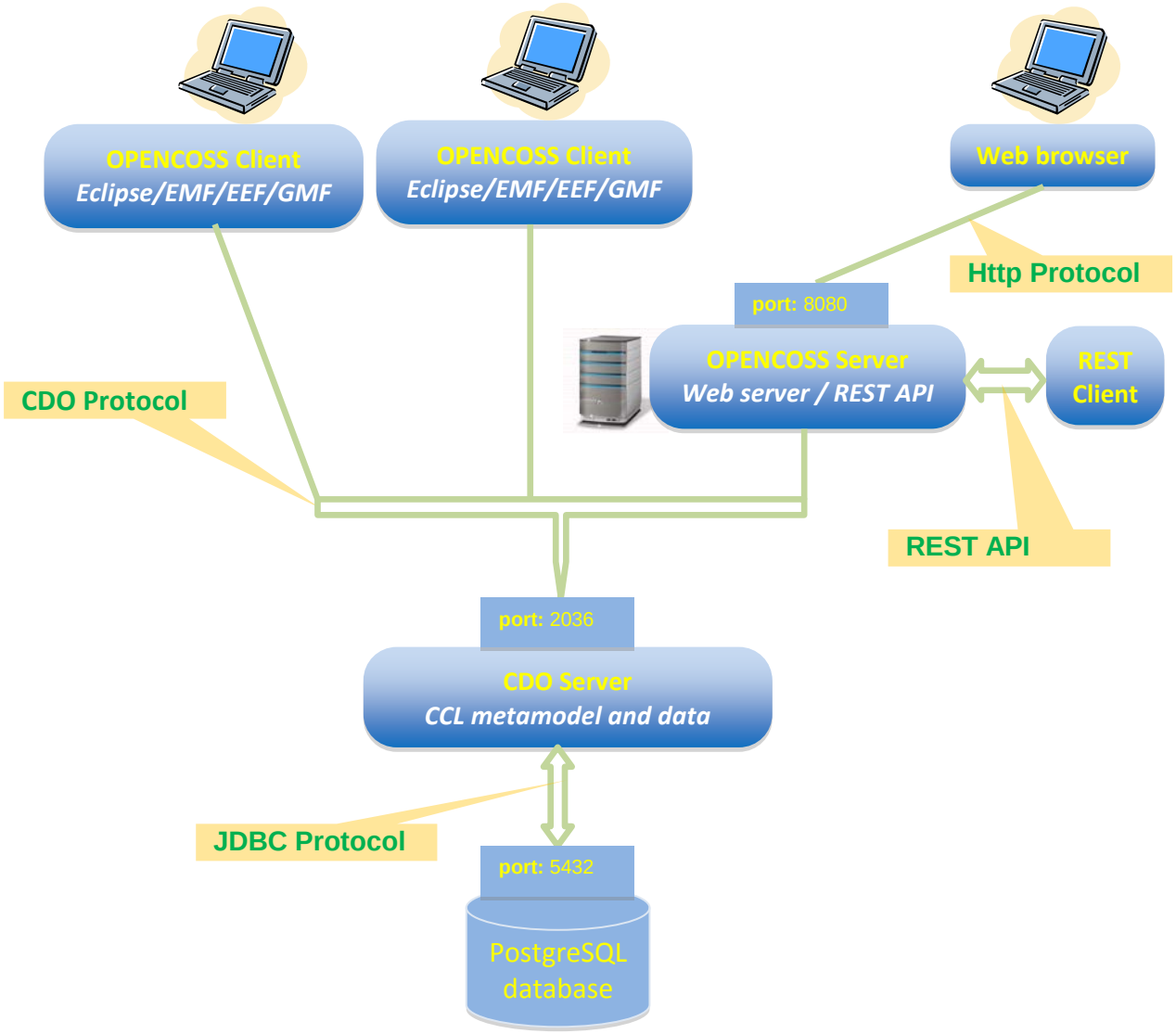


Figure 3. OPENCOSS platform tools - technologies and collaboration

Regarding the OPENCOSS clients, they provide:

- Editors for entire CCL model data, in particular for reference frameworks, assurance projects, argumentation, evidence chain, and process data.
- Evidence Change Impact Analysis engine.

The Functionality of the editors is described in detail in the user manual chapter (OPENCOSS Platform tool user manual). The client source code is described in the Technologies used in implementation of the clients chapter.

The OPENCOSS platform server has been implemented as a set of web applications using Apache Tomcat server. It is supposed to be installed on a corporate server machine. OPENCOSS platform server provides the following functionality:

- Host web reports, which provide analytical views of data stored in OPENCOSS repository.
- The web interface is accessible for users via their web browsers.

- Host optional services API, which allow to interface with OPENCOSS platform.

The functionality provided by the OPENCOSS server is described in the OPENCOSS Platform tool user manual chapter. Note that some of the service APIs can be hosted separately as satellite servers.

2.2 Source code description

This chapter presents the structure of the source code packages of both server and clients of the OPENCOSS platform implementation. This source code has been developed in the scope of T6.4 implementation effort. Both the client and the server have been implemented in Java language.

Various technology libraries have been used (see Technologies used in implementation of the server and Technologies used in implementation of the clients). The code has been committed to the OPENCOSS SVN source control repository. It is hosted at the following URL: https://svn.win.tue.nl/repos/opencoss-code

2.2.1 Server source code

The source code that implements the OPENCOSS platform server has been committed to the following location: *https://svn.win.tue.nl/repos/opencoss-code/trunk/common*. There are the following Java packages:

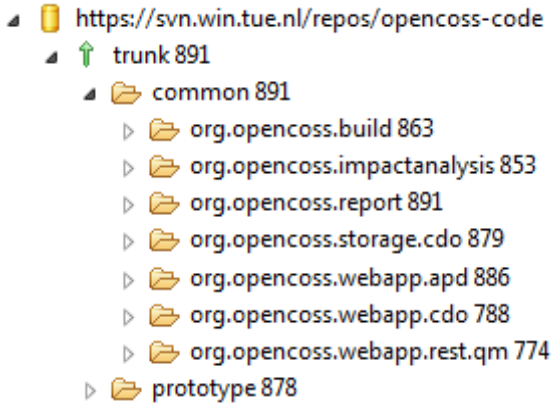


Figure 4: SVN structure

Most of the packages (all except *org.opencoss.webapp.apd*) play a direct or indirect role in evidence chain management implementation, which is the subject of this deliverable. The general description of these implementation packages is as follows.

org.opencoss.build - This package contains configuration files which are used for setting up a development environment and installation builds. Both topics are described in detail in the OPENCOSS Developer Manual (Developer manual - Server chapter).

org.opencoss.impactanalysis - This package contains the implementation of the change impact analysis module. This engine is used by OPENCOSS platform clients to call and execute change impact analysis. The algorithm logic and usage of this module have been described in detail in the Change Impact Analysis chapter.

org.opencoss.webapp.reports - This package contains the implementation of OPENCOSS platform server web pages. The functionality has been developed using Vaadin, which is a Java framework for building modern web application (*http://www.vaadin.com*).

The implementation of OPENCOSS platform web pages follows a Model-View-Controller (MVC) design pattern. The following packages play the most important parts in the implementation of the pattern:

- *Model* information is provided by *org.opencoss.webapp.reports.dao* package classes. These classes have been designed according to the Data Access Object (DAO) pattern, which uses CommonStorageProvider code to access the CDO repository of the OPENCOSS platform. CommonStorageProvider is described in the Common Storage Provider chapter.
- The *View* part is supported by *org.opencoss.webapp.reports.view* package. The classes there implement Vaadin panels and represent various parts of GUI web layout. A main class called by Vaadin framework is *OpencossApplication.java*, which provides a framework for the layout of OPENCOSS platform server web pages.
- The *Controller* part is implemented *by org.opencoss.webapp.reports.manager* package. When some *view* class is about to render any data element, it calls the *manager* class (which contains the business logic), reads data from *model* supported by DAO classes, performs any additional operations if necessary, and returns the prepared data to the *view* module.

org.opencoss.storage.cdo - This package contains classes fur using the CDO server in the OPENCOSS platform. This server provides a common storage for all OPENCOSS platform clients and a server. It accesses PostgreSQL database as its data backend. It is implemented in the *StandaloneCDOServer.java* class. The details of this piece of source code usage are described in the Common Storage Provider chapter. Additionally to common storage implementation, this package contains utility classes used when accessing the CDO server by its clients.

org.opencoss.webapp.cdo - This package constitutes a separate web application which is responsible for starting a CDO Server that handles data requests from OPENCOSS clients and a server. See the Common Storage Provider for a detailed explanation of how the CDO server gets started by this web application.

org.opencoss.webapp.rest.qm - The package contains a first approach for integration with evidence external tools - in this case QM. It provides a REST API that provides details about evidence pieces stored in the OPENCOSS platform database. This approach has been discontinued in the 3rd prototype. This aspects is discussed in detail in Evidence REST API and initial integration with QM.

org.opencoss.ws.evm - The package contains an approach for an integration with external safety assurance as medini analyze. It provides a generic CRUD based REST API without an actual implementation but delegate interfaces to separate server from management functionality.

org.opencoss.ws.evm.emf - The package contains an implementation of the evidence management interface – as defined by *org.opencoss.ws.evm* – for EMF repositories in general and CDO in special. Both packages are required to setup a CDO backend for an evidence management service. Automated test cases are written and available in *org.opencoss.ws.evm.test*.

2.2.2 Client plugin source code

The source code implementing the OPENCOSS platform clients has been committed to the following location: *https://svn.win.tue.nl/repos/opencoss-code/trunk/prototype/plugins*. The following Java packages are located there:

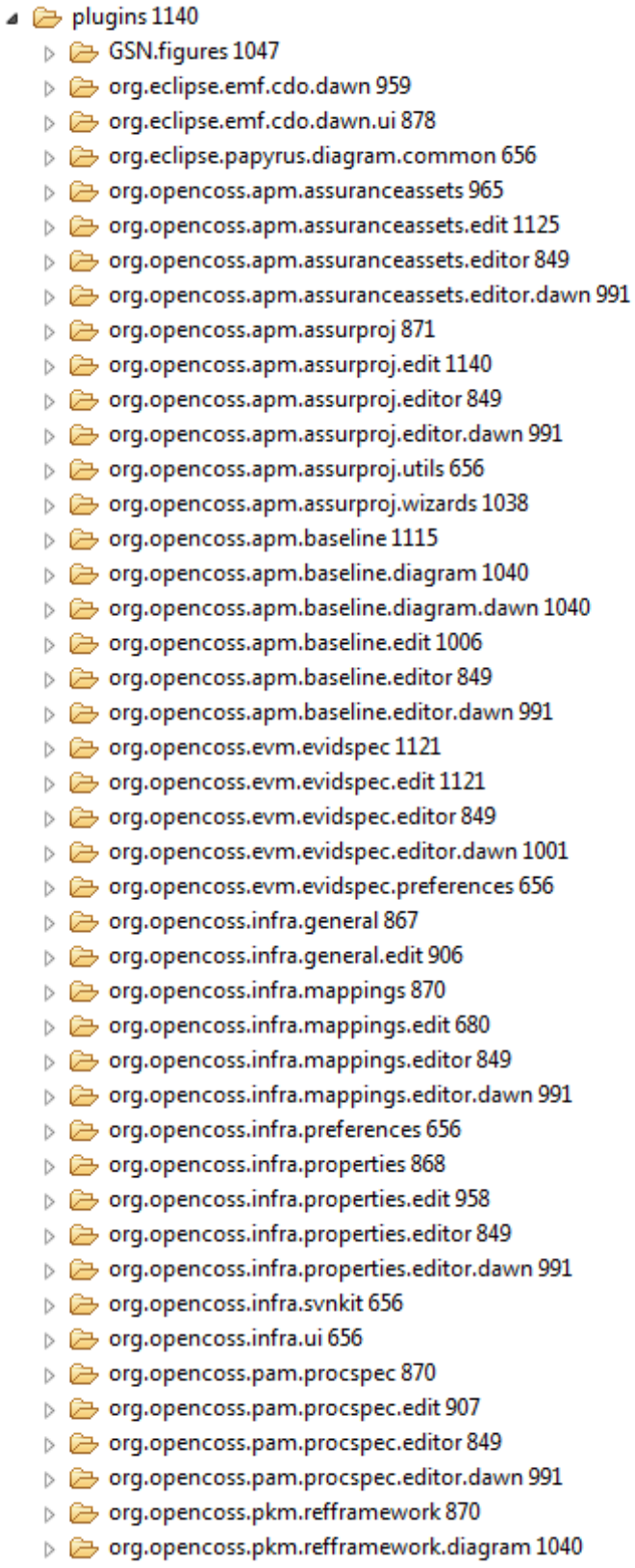


Figure 5: SVN structure plugin

A general description of these implementation packages is as follows.

org.opencoss.evm.evidspes - In this plugin, the evidence metamodel is defined and stored, and the Java implementation classes for this model are generated.

org.opencoss.evm.evidspes.edit - This plugin contains a provider to display evidence models in a user interface.

org.opencoss.evm.evidspes.editor - This plugin provides the user interface to view instances of the model using several common viewers, and to add, remove, cut, copy and paste model objects, or modify the objects in a standard property sheet.

org.opencoss.evm.evidspes.editor.dawn - This plugin is an extension of the previous one. It aims to communicate with the CDO Server to store the generated model in a database instead of a file.

org.opencoss.evm.evidspec.preferences - This plugin defines the default preferences for the communication with the SVN repository, thus it defines the type of repository (local or remote) and a user and password to connect with the remote repository.

org.opencoss.infra.svnkit - In this plugin, the functionalities necessary for the communication with the repository SVN (SVNKIT V1.3.8) are defined, to export and import artefacts.

org.opencoss.evm.evidspes.editor.dawn - This plugin is an extension of the previous one. It aims to communicate with the CDO Server to store the generated model in a database instead of a file.

In addition, these plugins are necessary to handle the evidence properties:

org.opencoss.infra.properties - This plugin contains the definition of the Property metamodel, and the Java implementation classes for this model.

org.opencoss.infra.properties.edit - As the edit plugin for evidence, this plugin contains a provider to display the model in a user interface.

org.opencoss.infra.properties.editor - As the edit plugin for evidence, this plugin is an editor to create and modify instances of the model.

2.3 Technologies used for implementing the OPENCOSS platform server

This chapter presents the software development technologies that have been used in the implementation of the OPENCOSS server stack.

2.3.1 Web server infrastructure

The OPENCOSS platform server runs on Apache Tomcat (http://tomcat.apache.org/). All the main services, including web pages, web-enabled APIs, and common data storage infrastructure, have been implemented as separate web applications deployed on the web server. The common storage infrastructure is presented in the CDO Server implementation chapter. How to build OPENCOSS server web applications is described in Building OPENCOSS server web application war files.

2.3.2 Integration layer

The integration layer is responsible for providing services to instantiate and bind the server code components together. This layer is intended to ease the burden of creating boiler-plate source code. In order to facilitate this, the Spring framework (http://spring.io/) is used. It is the most popular integration framework for Java, provides up-to-date solutions, and is actively developed. For example, the Spring framework is used in the following places in OPENCOSS server source code:

- In *org.opencoss.webapp.cdo* web application, in order to instantiate *StandaloneCDOServer* class when Apache Tomcat is started. It is configured in *webapp\WEB-INF\applicationContext.xml*:

  <bean class="org.opencoss.storage.cdo.StandaloneCDOServer">

  </bean>

- In DAO classes in *org.opencoss.webapp.reports.dao* package - see Server source code for DAO classes description. "*@Component*" Spring annotations are used in order to mark the appropriate classes to be instantiated by the Spring framework. "*@Autowired*" annotations are used in *org.opencoss.webapp.reports.manager* classes so that DAO objects instances are injected in the proper places in manager classes. For example, *BaseActivityManager.java* contains the following Spring invocations*:*

  @Component

  public class BaseActivityManager {

  public static final String SPRING_NAME = "baseActivityManager";;

  @Autowired

  private BaseActivityDAO baseActivityDAO;

  …

  }

2.3.3 Communication layer for exposing web-enabled APIs

Apache CXF framework (http://cxf.apache.org) has been used in OPENCOSS platform server as a technology for web API implementation. This framework provides support for the two most popular API protocols in web communication:

- Full-blown XML-based web services based on SOAP and WSDL, driven by JAX-WS
- A lightweight approach of RESTful web services, driven by JAX-RS

These standards are well established, built upon other widely adopted standards like JAXB.

The 3rd prototype implementation contains an initial approach of the API to integrate with QM to illustrate approach that can be followed to make such an integration. It has been implemented in *org.opencoss.webapp.rest.qm.service* package and described in the Evidence REST API and initial integration with QM chapter.

Hereafter there is an example of source code exposing REST services implemented in *ArtefactService.java* class. CXF framework annotations, which specify the API, have been bolded below.

```
@GET
@Description(value = "Returns a list of ALL Artefacts")
@Produces({MediaType.APPLICATION_JSON})
public Response getArtefacts(@QueryParam(JsonOption.INDENT_PARAM)
        @DefaultValue(JsonOption.FALSE) boolean indent)
{
…
}
```

2.3.4 Web framework

Web Graphical User Interface in OPENCOSS web server has been developed using the Vaadin framework (https://vaadin.com). This library provides a rich set of widgets, and supports model-view-presenter pattern to allow well-structured source code development. The source code structure of this framework usage is described in the Server source code chapter above.

The main class that implements core web page UI and integrates all the graphical panels has been implemented in the *org.opencoss.webapp.reports.webapp.OpencossApplication* java class.

2.4 Technologies used in implementation of the clients

The technologies involved in the generation of the Evidence editor are EMF, EEF and Dawn.

EMF generates the necessary code to manage the Evidence meta-model part of the CCL and also generates a basic editor. The appearance of this editor can be seen in the screenshot below.

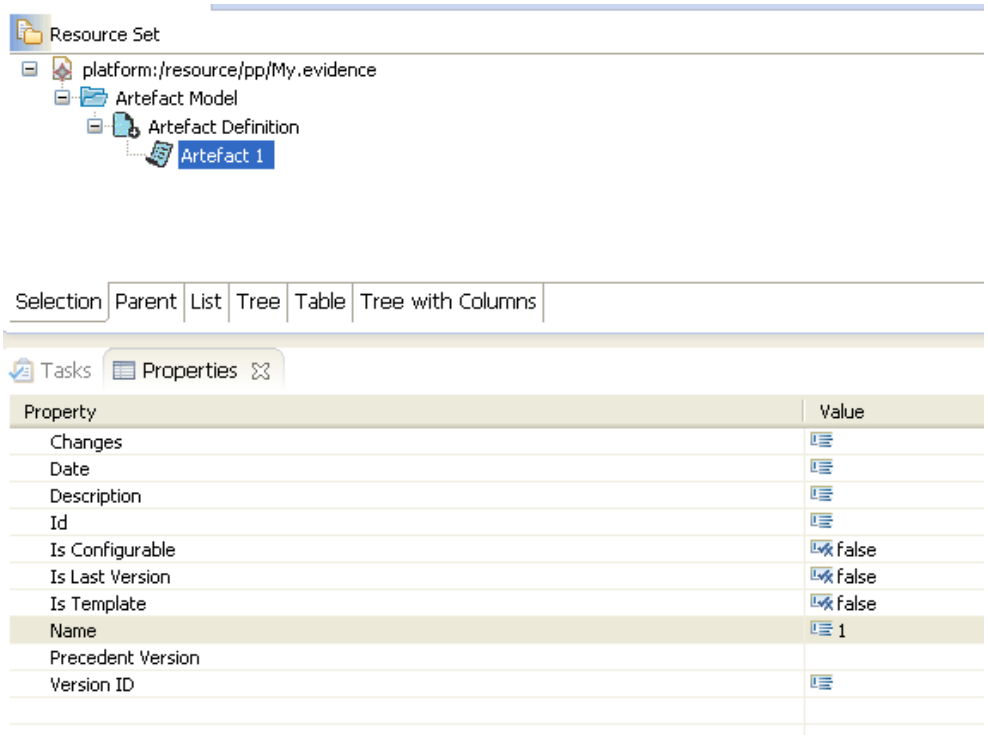


Figure 6. Example of OPENCOSS client GUI which uses EMF

EEF generates code to improve the graphical look and the usability of the EMF editor (see figure below). This editor has more easy-to-use controls to specify the different properties of the entities included the evidence metamodel and the relations between them, and it has the possibility of grouping the properties in tabs.

The code of this editor has been modified to integrate it with SVN as evidence repository and to capture the modified artefacts in order to call the Impact Analysis engine explained in the Change Impact Analysis chapter.

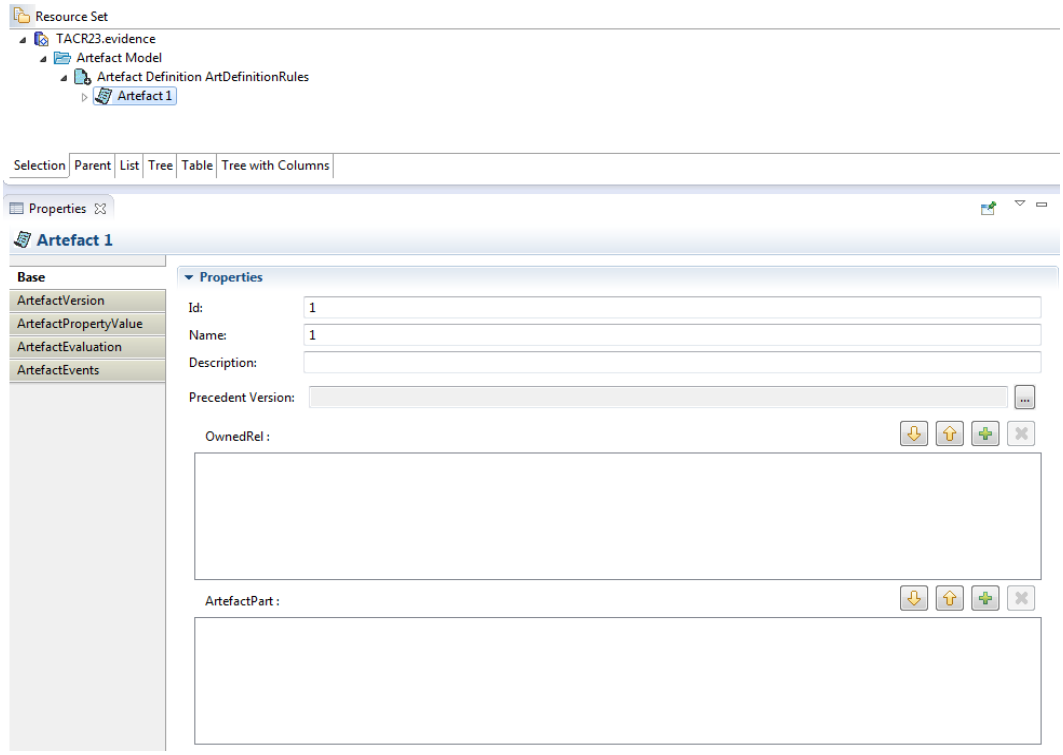


Figure 7. Example of OPENCOSS client GUI which uses EEF

Dawn framework builds a bridge between the EEF Editor and the CDO server allowing the storage of the generated evidence models in the database.

2.5 CDO Server implementation

As introduced in the previous chapters, CDO is a Java model persistence solution for data models. In OPENCOSS platform tools, both the clients and the server use a common data storage, which has been implemented based on CDO.

The following subchapters describe why CDO persistence solution has been chosen, how CDO common server is used in OPENCOSS platform implementation, and how it is accessed by clients.

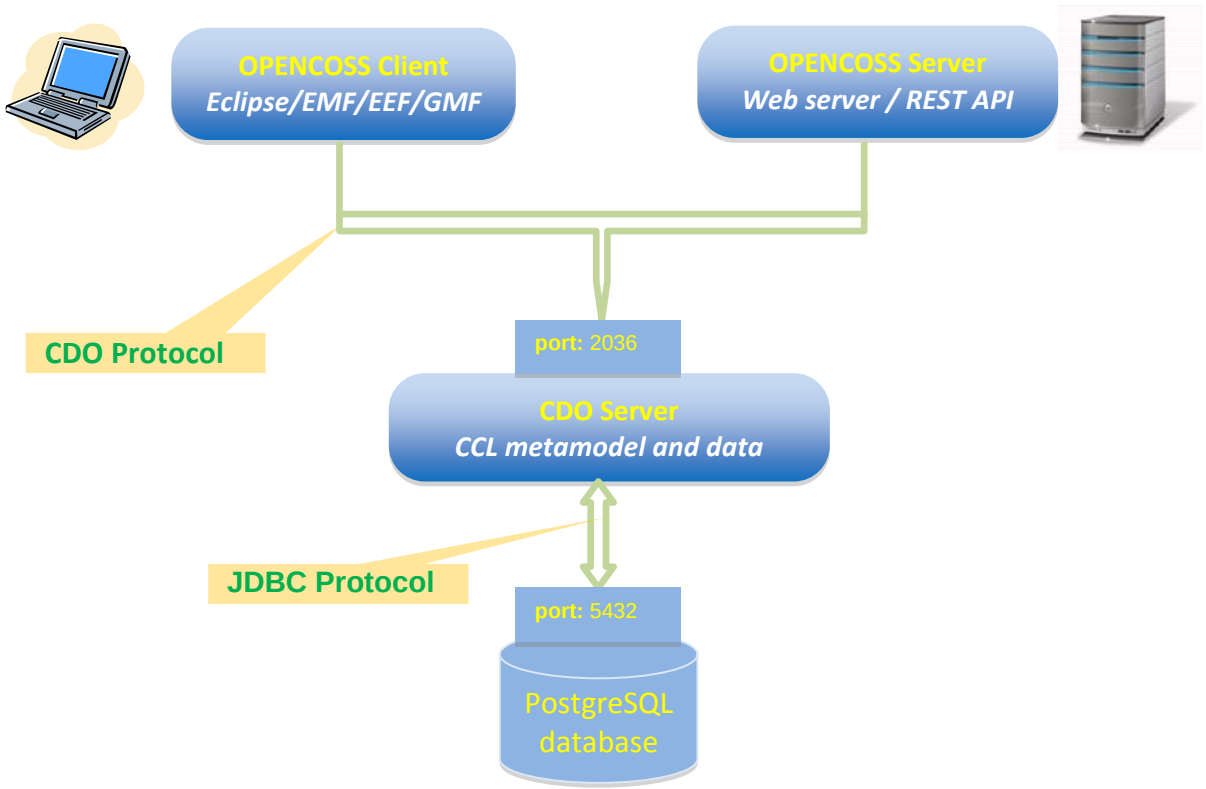


Figure 8. Communication between OPENCOSS platform client, server and data storage

2.5.1 Teneo vs CDO

At the beginning of T6.4, research on EMF-based storage technologies was performed. Three solutions were investigated: CDO, Teneo and EMFStore. Results of this evaluation have been placed in *WP6/D6.5_in_progress/TechnologyResearch/CDOvsTeneovsTexovsEMFStore* folder in OPENCOSS project source repository.

As described in D6.5, chapter "2.1 Common Storage Provider", the first approach of data persistency implementation for OPENCOSS platform was based on Teneo technology, as it was a reasonable recommendation after the evaluation of the three storage technologies. This solution worked well on OPENCOSS server side. However, as the implementation progressed on OPENCOSS Eclipse clients side, serious obstacles arose. It turned out that integration of Teneo storage with GMF (Graphical modelling Framework), one of the technologies used in OPENCOSS clients tools, is problematic. Furthermore, there was very little information and community support on the Internet for such configuration. It was decided to evaluate GMF with CDO storage, which ranked second in our technology evaluation. The integration proved to work reasonably well, thus CDO technology has been chosen to implement OPENCOSS platform storage compatible with EMF models.

2.5.2 CDO server implementation

Initialization of CDO server storage for OPENCOSS platform has been implemented in the *StandaloneCDOServer.java* class. The CDO server is started together with OPENCOSS server Apache Tomcat. There can be various OPENCOSS server web applications deployed on Apache Tomcat. For the 2nd prototype, *org.opencoss.webapp.reports* webapp facilitates OPENCOSS web reports and *org.opencoss.webapp.cdo* application runs the CDO server itself. The *org.opencoss.webapp.cdo* web application uses Java Spring framework, and based on the configuration specified in *webapp\WEB-INF\applicationContext.xml*, it instantiates *StandaloneCDOServer* class when Apache Tomcat is started.

```
<bean class="org.opencoss.storage.cdo.StandaloneCDOServer">
</bean>
```

This constructor of the class initializes the CDO server for OPENCOSS, i.e.:

- It connects to PostreSQL database, which is a backend with actual data storage
- It sets up CDO server and starts to listen to requests from CDO clients on the preconfigured port

The settings, both of PostgreSQL database and CDO port can be set up in *opencoss-properties.xml* configuration file. The file is read from the operating system's user home directory and has the following structure:

```
<properties>
    <entry key="dbHost">10.9.1.25</entry>
    <entry key="dbPort">5432</entry>
    <entry key="dbName">cdo-opencoss</entry>
    <entry key="dbUser">opencossdbms</entry>
    <entry key="dbPassword">opencossdbms</entry>
    <entry key="serverAddress">localhost:2036</entry>
</properties>
```

The details of the file configuration have been described in OPENCOSS Platform tool user manual chapter.

2.5.3 Accessing the CDO server from source code

This chapter briefly describes the source code used to access the CDO server from client side, both from OPENCOSS clients and a server.

2.5.3.1 Accessing the CDO server from OPENCOSS server source code

As described in previous chapters, any piece of source code that accesses OPENCOSS data storage, i.e. the CDO server, has been designed to follow a DAO (Data Access Object) approach. The implementation of DAO classes has been placed in *org.opencoss.webapp.reports.dao* package.

When any DAO class needs to access CDO storage, it calls *StandaloneCDOAccessor.java* object, which implements a common way of connecting to OPENCOSS data repository. The constructor of this class initiates a CDO connection to the server.

Similarly as in the case of the CDO server, the connection settings are read from *opencoss-properties.xml* configuration file.

2.5.3.2 Accessing the CDO server from OPENCOSS client code

OPENCOSS clients, which have been implemented as Eclipse plugins and can run on any user machine, have a separate configuration, which allows user to specify CDO server connection settings. It is accessible for users in Eclipse Preferences menu.

In clients' source code, the following CDO library invocation is used in order to connect to the CDO server:

```
org.eclipse.emf.cdo.dawn.util.connection.CDOConnectionUtil.instance.init(
                                    PreferenceConstants.getRepositoryName(),
                                    PreferenceConstants.getProtocol(),
                                    PreferenceConstants.getServerName());
                CDOConnectionUtil.instance.openSession();
```

2.6 Evidence REST API and initial integration with QM

2.6.1 Overview

The *org.opencoss.rest.qm* module is responsible for integrating external tools within the OPENCOSS platform to provide additional services to the platform. The initial goal of such integration was to allow non-java software modules to provide some of the OPENCOSS functionalities by using a specific REST API to browse and update the objects manipulated by the OPENCOSS platform. As an example of such integration, QM was chosen as a candidate technology to provide the impact analysis functionality of the OPENCOSS platform.

QM is an artefact manager tool that uses a certification project model to access artefacts of a safety project, to construct traceability links, and from that information to provide several functionalities to help in the certification process (impact analysis, documentation aggregation, and artefact life cycle management).

2.6.2 Integration details

In order to integrate an external tool for such a central functionality, the tool needs to have access to the complete database of artefacts and traceability links (e.g. Artefact relationships database).

To do so, several REST functions are provided:

- *ArtefactService/getArtefacts*, returning the complete list of artefacts
- *ArtefactService/{uid}*, returning the details on a single artefact
- *ArtefactService /{uid}/artefactParts*, returning the sub-artefacts
- *ArtefactService/{uid}/artefactRels*, returning the list of artefact relationships
- *ArtefactRelService*, returning the complete list of artefact relationships

On QM side, we used two different mechanisms to perform the integration:

- QM abstracts the means of accessing resources by using a concept of 'location'. So we implemented a new class of locations that use the above REST methods to give it access to the OPENCOSS-managed artefacts.
- To handle traceability links, we could also import the OPENCOSS artefact relationship database in the form of "traceability matrix" that is then used by QM to instantiate its own internal links.

2.6.3 Results

This integration has finally been cancelled for several reasons:

- The amount of messages that needed to be defined in order to provide a proper impact analysis was very large, and implementing all of them would have taken a significant time, greater than implementing the impact analysis functionality directly.
- The performance of such integration is also not great. In particular, for such a feature, the external tool needs to access all objects of the OPENCOSS platform, as well as their relationships. This leads to an almost complete duplication of the data between the OPENCOSS platform and the tool.
- Finally, such internal feature is really dependent of the CCL language and integrating a non-CCL tool leads to lowered functionalities, as there cannot be an exact match between this tool's data and functionalities with the data stored as CCL.

2.7 Evidence REST API and integration with medini analyze

2.7.1 Overview

Another potential use case for offering an API to external programs is the integration with existing safety assurance tools, i.e. tools used by safety analysts and engineers to perform safety analysis as FMEA, FTA, HARA or any other kind of method proposed by safety standards. The goal of such integration is to allow other tools that are not based on CCL or OPENCOSS to export their artefacts to the OPENCOSS platform in a uniform and cross-platform manner by using a dedicated REST API to create, read, update and delete (CRUD) artefacts in the OPENCOSS platform. The medini analyse tool – a commercial model based tool that supports multiple safety analysis techniques – was used as a proof of concept integration.

2.7.2 Integration details

In order to export artefact data from the external tool, the REST API has to offer typical CRUD (Create, Read, Update and Delete) style operations for the complete CCL Artefact meta-model plus a small set of navigation and exploration features. E.g. the tool is typically used by a safety

engineer that is working for a certain project, hence has to export the artefacts to the right assurance project in the OPENCOSS platform. All meta-information about an artefact (technique, version, description, additional properties etc. are exported by the tool. However, the real data are kept in the domain of the tool and need to be referenced from the platform data. This a tricky part of the integration because there is no common way of tool data storage. Some tools use proprietary databases where they store the data, others just store all data in local files. In both cases it is difficult to store a reliable location of the data to the OPNCOSS server. The OPENCOSSs platform is currently prepared to either store references to Subversion entries or an arbitrary URL not further analysed by OPENCOSS. If the tool does not support Subversion, a proprietary URL has to be encoded and stored that later can be used to find and open the document resp. data.

The medini tool supports multiple different working styles. The user may use Subversion to store its project data but is not forced to do so. He may also archive the complete project as a ZIP file and store it somewhere else on its own. Furthermore the tool is a rich client tool that is working solely on local files. Creating a reliable URL to address safety evidence data is not possible without using a central storage for these files and address them. To cope with the situation a small REST service as also implemented on the tool side to provide reverse lookup of artefact URLs in the tool.

2.7.3 Results

Due to cancelation of the first integration approach, the development of the second one did start rather late. The service uses the same techniques as the other REST APIS as mentioned before. However, to keep it lightweight and to support other integration scenarios, there is no dependency to any web-server or bean framework. Instead a very slim service module was implemented that can be launched as a separate service executable and that can be attached to either the OPENCOSS central server or to any in-memory artefact resource. That way the pure REST service in independent from the artefact management which can be the OPENCOSS platform but in turn also an arbitrary other artefact management tool. A test driven development style helped to write test cases against the REST API at the same time as implementing the functionality behind the service. The implementation was split into three plugins, one base plugin defining the API and data objects, usable by server as well as client side. Another plugin that implements the management part and that can be attached to the OPENCOSS central server or to an in-memory volatile data storage. The third and last plugin provides API and integration tests.

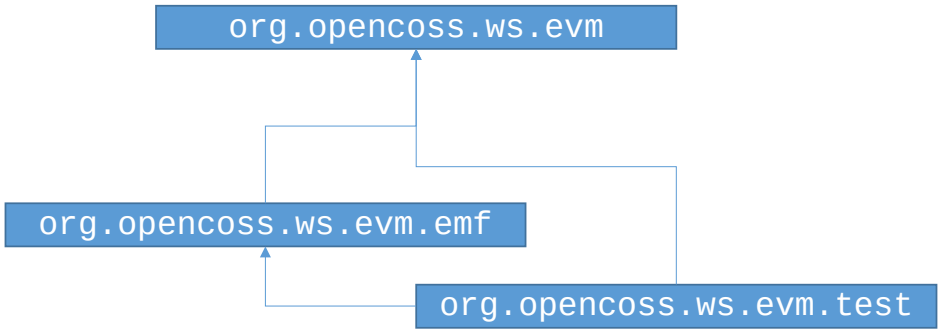


Figure 9. Evidence REST API bundles

The service was tested at API level using plain browser tools. For that a browser as Firefox is sufficient. In order to send correct REST service request, typically Addons as "RESTClient" arre required but can be easily installed via the Browser extension mechanism. In general the API accepts four kinds of HTTP requests, GET to read from the service, POST to create something new on the server, PUT to update existing elements and DELETE to remove an element on the server. Other methods are not supported. The method needs to be set in the REST Client before actually sending a request. The service only accepts (and sends) JSON as content type (except for some convenient calls as “ping”). The following picture shows an excerpt of supported API calls, the required URL path and the expected resp. the replied content types.

API Call	URL Path	Method	Accept	Send	Return
ping	/evidences/ping	GET	any	none	HTML, JSON or plain text
list artefacts	/evidences/artefacts	GET	JSON	none	JSON
list artefact Ids	/evidences/artefact-ids	GET	JSON	none	JSON
create artefact	/evidences/artefacts	POST	JSON	JSON	JSON
delete artefact	/evidences/artefacts/{id}	DELETE	any	none	none
update artefact	/evidences/artefacts/{id}	PUT	JSON	JSON	JSON
list definitions	/evidences/definitions	GET	any	none	JSON
list definitions ids	/evidences/definition-ids	GET	any	none	JSON

Figure 10. Allowed evidence REST API method calls and data formats (excerpt)

In theory the complete API can be covered using a plain browser + REST Addons, even though this is not a native “safety assurance” or “evidence management” tool. However, using the RETS API it would be possible in the future to write browser based applications too. The picture below shows the result of a “list artefacts” call.

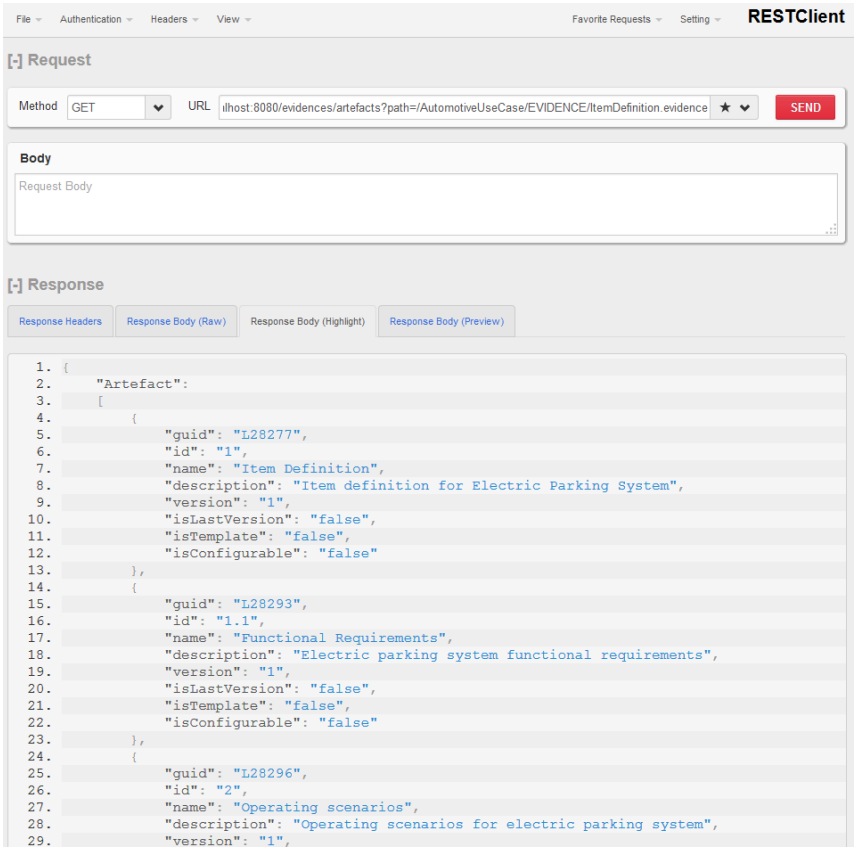


Figure 11. Test call result against REST API using plain browser

One big advantage of REST and data formats as JSON is the wide acceptance of both techniques and thus the availability of good and stable frameworks. The safety assurance tool medini analyse is an eclipse RCP based application. For Java stable libraries as Jackson & Jettison (JSON Marshalling), Jersey (REST Client & Server), and Jetty (HTTP Server) do exist and are widely used. Good documentation and examples are available. Based on these libraries, a proof of concept integration was developed as an add-on to the commercial safety tool medini analyse. The proof of concept implementation is fully based on the assumed integration scenario for OSLC described in chapter 5.3.3: safety engineers that are working with medini analyse are executing safety analysis and want to export the results of these analysis as artefacts to the OPENCOSS platform. The tool supports several different safety analysis techniques as FTA, FMEA, HARA, HAZOP and some more. From the viewpoint of OPENCOSS they are all treated similar with the only difference that different techniques were used.

The proof of concept implementation comes with an additional eclipse exporter that allows a user to export or update arbitrary artefacts to the platform. The tools connects to the server using the REST API and synchronizes all available artefacts in the user's tool workspace – selected or child of the selection – with artefacts in the OC platform. Artefacts that are newly created can be exported the first time. Artefacts that have been already exported in the past can be updated, i.e. a new version of the artefact is created in the platform.

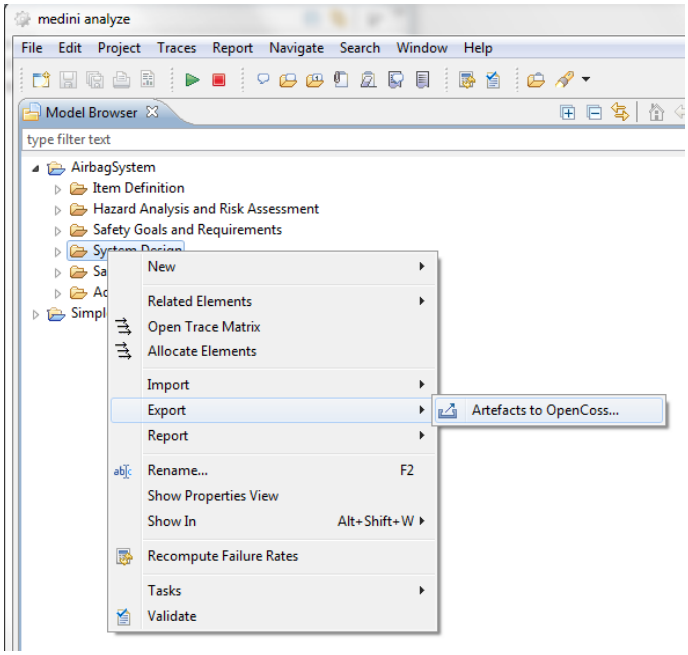


Figure 12. medini analyze exporter add-in using REST API

An expert view lets the user see the impact of the synchronization before actually executing it. New artefacts (artefacts that were never exported before) are marked as additions while already exported artefacts are marked as updates mixed with information of current version available on the server side (e.g. last updated current version etc.). The user is able to make a final decision upon which artefact he ultimately wants to export to the OC repository.

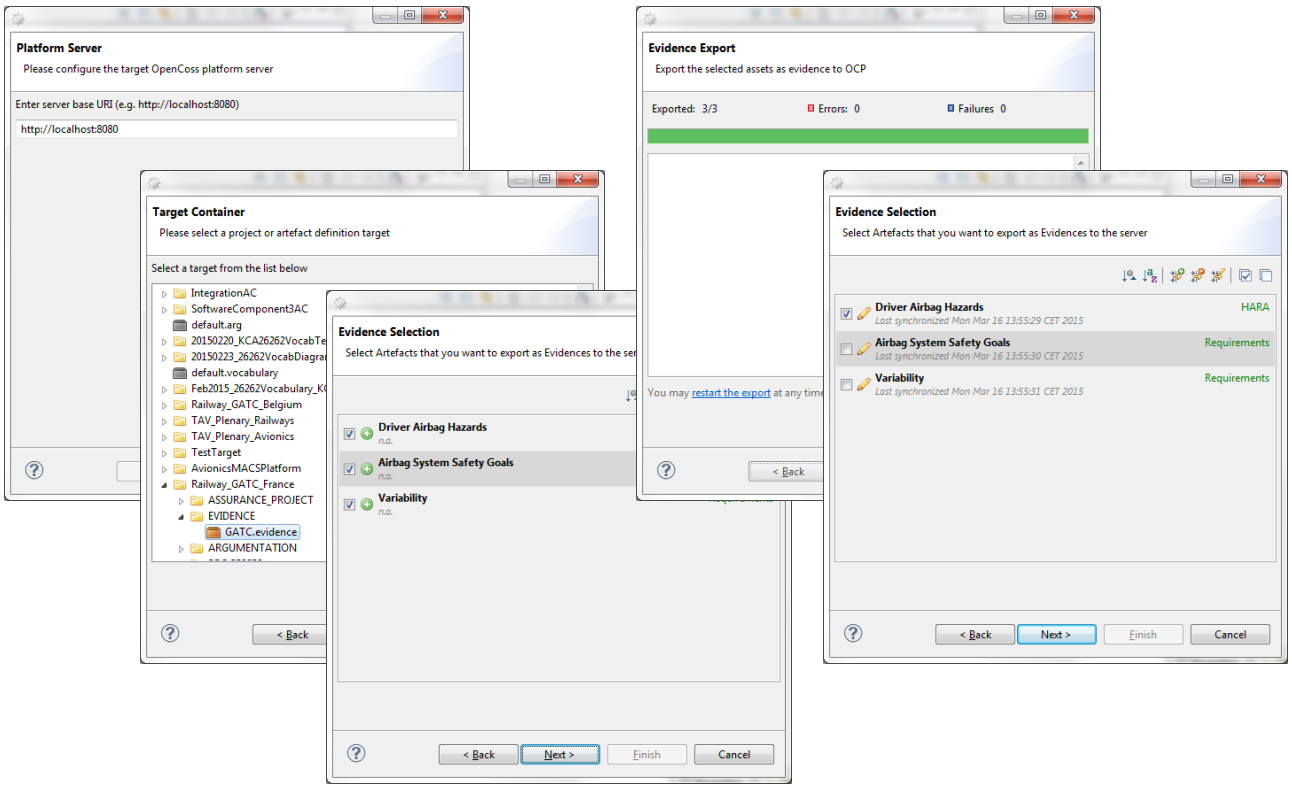


Figure 13. Artefact export wizard add-in for medini analyze

After synchronization, required meta data about the artefact GUID, about were artefacts was exported, and when the last update was done is stored internally in medini analyse as annotations to the safety models. The information are used on later synchronization to keep a 1:1 trace relation with the artefacts on the server.

2.8 Change Impact Analysis

Change Impact Analysis (IA) logic has been developed in *org.opencoss.impactanalysis* Java package. The implementation classes provide an impact analysis engine, which can be used in any place of OPENCOSS source code. Currently, the engine is used in OPENCOSS client Eclipse plugins in Evidence Editor functionality.

Hereafter there is a detailed description presenting how OPENCOSS Change Impact Analysis engine works and how to use it in a source code.

- *ImpactAnalyser impactAnalyser = new ImpactAnalyser();*
  The above code instantiates ImpactAnalyzer class which contains IA engine's code.
- Let us assume that user modifies some artefact (which has *theArtefactCODId* id) using OPENCOSS client editor. Upon the modification, a CCL AssuranceAssetEvent with *EventKind.MODIFICATION* is assigned to the changed Artefact.

- Now, the change impact analysis engine can be run, in order to detect what kind of effect the change has on the related artefacts. In order to run IA in the source code, the following code can be issued:

  List<ArtefactsRelationImpact> artefactsRelationImpacts =

  impactAnalyser.listArtefactsRelationImpacts(theArtefactCDOId, EventKind.MODIFICATION));

  listArtefactsRelationImpacts() method takes the following parameters:
  - *artefactCDOId* - it is an Id of the artefact object from which the IA should be started
  - *eventKind* - the kind of event for which the analysis should start on the initial artefact. Two basic values of *EventType* taking part in Impact Analysis are *EventKind.MODIFICATION* and *EventKind.REVOCATION.*

  In our example, we are passing *EventKind.MODIFICATION,* because the artefact has been changed and we want to analyse this modification effect.

- The change impact analysis algorithm implemented in *listArtefactsRelationImpacts()* is described below.

  The main pieces of information used by the IA engine are relations between Artefacts objects stored in *ArtefactRel* CCL entity.

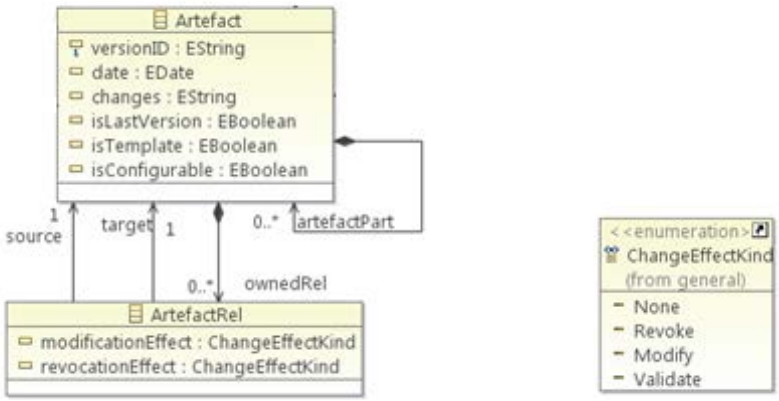


Figure 14. Artefact metamodel

Two Artefacts are considered related when there is an *ArtefactRel* instance pointing to one of them as a source and another of them as a target. Please note that *ArtefactRel* has *modificationEffect* and *revocationEffect* attributes.

Note: An ArtefactRel object for specific two artefacts can be added in the following ways:
- A user can add this entity manually in the Evidence Editor of OPENCOSS platform client
- ArtefactRel entity is added automatically when a parent-child relation is established between two artefacts. When adding artefactPart to parentArtefact, a new ArtefactRel object is created, with modificationEffect=MODIFY and revocationEffect=MODIFY, *source* pointing to parentArtefact and *target* to artefactPart.

It has been arranged that a direction of analysis flow is the following: *ArtefactRel* "*target* affects the *source*". When impact analysis is started:

- It starts from *artefactCDOId* for the specific *EventKind* (either *Modify* or *Revoke*),
- It looks into the related *ArtefactRel* (for which the *artefactCDOId* is a *target*) object
- It traverses to the artefact pointed by *ArtefactRel source*
- Depending on the initial *EventKind* (either *Modify* or *Revoke*), it takes the value of *modificationEffect* or *revocationEffect* from the *ArtefactRel* and assumes the appropriate *AssuranceAssetEvent* on the reached source artefact.

For example, let's assume that there are the following *Artefact* and *ArtefactRel* dependencies:

ArtefactA ---- ArtefactRelA(ModificationEffect:MODIFY, RevocationEffect:REVOKE) ---> ArtefactB

ArtefactB ---- ArtefactRelB(ModificationEffect:REVOKE, RevocationEffect:REVOKE) ---> ArtefactC

ArtefactC ---- ArtefactRelC(ModificationEffect:MODIFY, RevocationEffect:VALIDATE) ---> ArtefactD

ArtefactD ---- ArtefactRelD(ModificationEffect:MODIFY, RevocationEffect:REVOKE) ---> ArtefactE

- The engine starts with *EventKind.MODIFICATION* for *ArtefactA* and navigates via *ArtefactRelA* to *ArtefactB* and because *ArtefactRelA:ModificationEffect* equals *MODIFY*, it reaches *ArtefactB* with *EventKind.MODIFICATION* change effect.
  Note: this change effect event is not saved in storage yet. Now it is only used for further traversal, and will be returned as part of the result of *listArtefactsRelationImpacts()* method.
- The engine continues from ArtefactB with EventKind.MODIFICATION and navigates to ArtefactC and because *ArtefactRelB:ModificationEffect* equals REVOKE, it reaches *ArtefactC* with *EventKind.REVOKE* change effect.
- The engine continues from *ArtefactC* with *EventKind.REVOCATION* and traversal path ends here because of *ArtefactRelC:RevocationEffect* equals *VALIDATE*.

- Change effects of the above traversal are returned by *listArtefactsRelationImpacts* method as a *List<ArtefactsRelationImpact>* result object which contains the following information:

  ArtefactRelationImpact {

  int recursionDepth,

  ArtefactRel originalArtefactRel,

  ModificationEffect impactedModificationEffect

  }

  originalArtefactRel - is an original *ArtefactRel* information, containing relation traversal *ModificationEffect* and *target* and *source* Artefacts.

  impactedModificationEffect - is the exact change effect detected by IA path traversal.

  In our example, the result would be the following:

  (1, ArtefactRelA, ModificationEffect.MODIFY)

  (2, ArtefactRelB, ModificationEffect.REVOKE)

- The result of the above analysis can be displayed using the below code.

String result = impactAnalyser.visualizeArtefactsRelationImpacts(artefactsRelationImpacts);

For the prototype implementation, the result visualization has been implemented as a simple textual output. It can be further extended in the future.

- User observes the IA result output and can take one of two optional actions:
  - Abandon the artefact change. In this case the algorithm ends here.
  - Commit *ArtefactA* change and execute all change effects detected by the impact analysis and returned as *List<ArtefactsRelationImpact>*
- In order to execute all the actions detected by change impact analysis, the following code can be called

impactAnalyser.executeArtefactsRelationImpacts(artefactsRelationImpacts);

As a result of this invocation, *AssuranceAssetEvent* items are added to the artefacts along the impact analysis result path.

Above algorithm affects the Artefact lifecycle. States of this lifecycle are presented on the figure below.

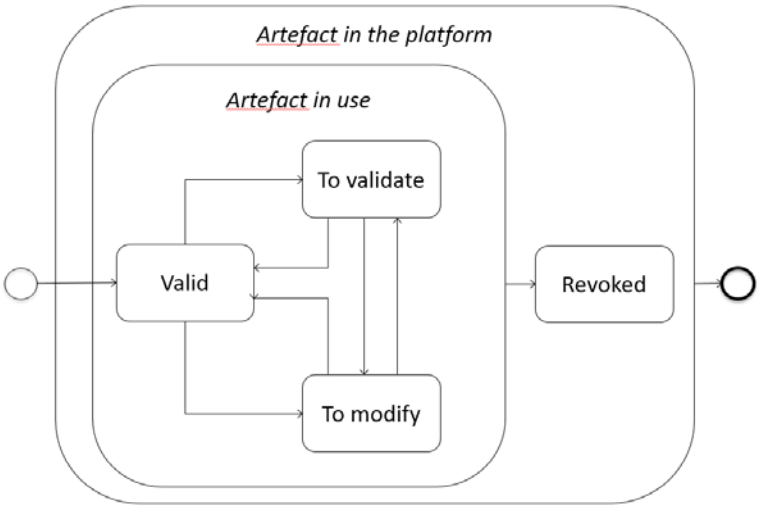


Figure 15. Artefact lifecycle from the IA point of view

Some of these states require action from user – like “To validate” and “To modify”. To address some restrictions of CCL these two states are recognized by the presence of given event date or lack of it. This signals that action from the user is required and after this action date of the event is set.

IA-induced user actions that need to be performed in the assurance project are presented on Compliance Estimation Report and Compliance Report. These reports are described in work package 7 in details. A screenshot of the web interface is presented below to show situation,

when “To validate” and “To modify” action are required from the user as a result of IA execution. User simply clicks “Modified” and “Validated” action buttons to report performed action.

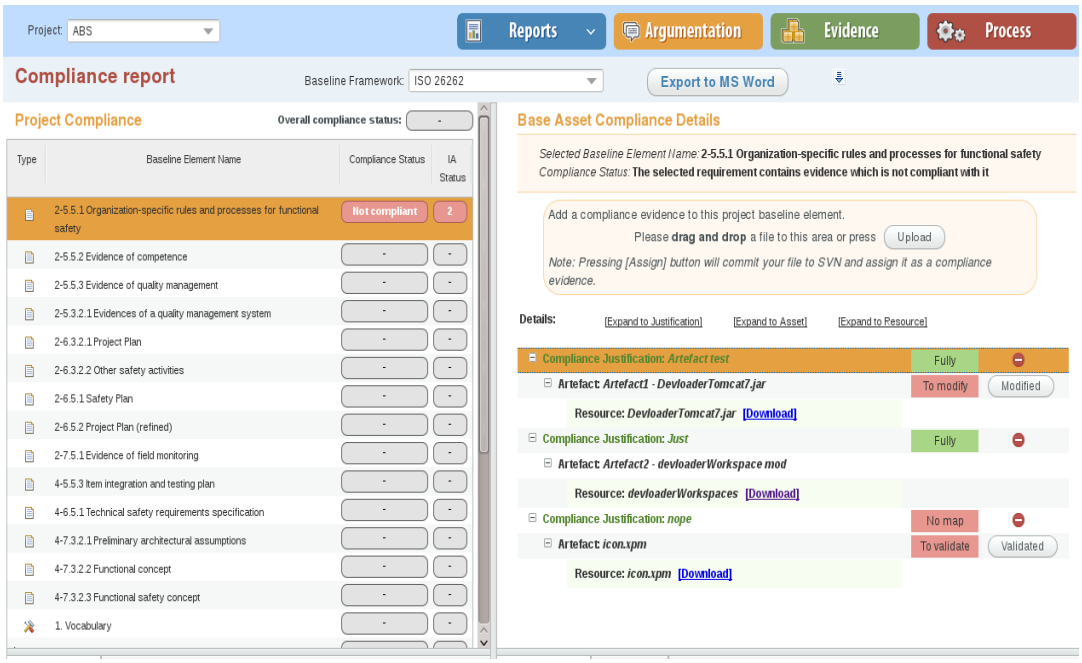


Figure 16. Web interface showing two IA-induced actions required to be taken by user

3 OPENCOSS Platform tool user manual

The manual describing how to use all the functionalities implemented in the OPENCOSS platform can be found in the OPENCOSS SVN repository at:

https://svn.win.tue.nl/repos/opencoss/WP-transversal/Implementation/ThirdPrototype/OPENCOSS_Prototype3_UserManual.doc

4 OPENCOSS Platform tool developer manual

This chapter provides detailed instructions for software developers. It explains how to install and set up the development infrastructure for OPENCOSS platform tools, and how to run the tools in the development mode. Client and server environments are presented in the chapters below.

4.1 Developer manual - Server

This chapter contains instructions how to set up OPENCOSS platform server development and build environments. In particular it describes:

- Installation of OPENCOSS platform database
- Setting up Eclipse IDE, source code checkout, and compilation
- Running OPENCOSS server in Eclipse IDE debugger
- Building OPENCOSS server web application war files

4.1.1 Installation of OPENCOSS platform database

4.1.1.1 Installation of PostgreSQL database

OPENCOSS platform tools use PostgreSQL database. The detailed description of how to download and install PostgreSQL database server has been provided in OPENCOSS Platform tool user manual, in the "Installation of PostgreSQL database" subchapter. Please follow the instructions presented there.

4.1.1.2 Creating OPENCOSS database in PostgreSQL

After PostgreSQL server has been created, a database for OPENCOSS tables should be prepared. Please follow the instructions from OPENCOSS Platform tool user manual, "Creating OPENCOSS database in PostgreSQL" chapter.

4.1.2 Installation and setup of Eclipse IDE

This chapter presents how to install and configure Eclipse IDE for OPENCOSS development. Checking out the code from source code repository is also described.

4.1.2.1 Installation of Java

Java JDK 8 (Java Development Kit) is required in order to compile and run OPENCOSS server from the source code. Having JRE (Java Runtime Environment) only is not enough. In order to check if you have JDK installed, please open a command prompt and type:

```
> javac -version
```

If JDK has been installed, *javac -version* output, eg. `javac 1.8.0_22`, will be presented. In case javac is not a recognized command, please download JDK 8 and install it from http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html

4.1.2.2 Installation of Eclipse IDE

Eclipse IDE is needed to work with OPENCOSS source code. The instructions for downloading and configuring it are as follows:

1. Download the latest version of Eclipse IDE for Java EE Developers from http://www.eclipse.org/downloads/.
2. Install it by following the instructions provided on the web site.
3. Run the Eclipse IDE pointing to a new empty workspace folder.
4. So that your Eclipse environment uses all the required libraries (e.g. EMF, EEF, GMF, CDO) necessary for OPECOSS source code, please set up your Eclipse to use the **target platform**, i.e.:
   - Download OPENCOSS Eclipse target platform bundle from the following location (depending on your platform):
     http://77.252.162.49:8080/opencoss/targetPlatform/20140227/OpencossTargetPlatform_20140227_Win_x64_bundle.zip
     http://77.252.162.49:8080/opencoss/targetPlatform/20140227/OpencossTargetPlatform_20140227_Win_32_bundle.zip
     http://77.252.162.49:8080/opencoss/targetPlatform/20140227/OpencossTargetPlatform_20140227_Linux_x64_bundle.tar.gz
   - Unzip the package into your target folder
   - Configure your Eclipse to use the target platform which has just been downloaded. In order to do this:
     - Go to the unpacked target platform package and open *"How to use OPENCOSS Eclipse Target Platform.pdf"* document which is inside.

     Follow the instruction from this document starting from step 2.2 there: *"2.2 Configure target platform in your fresh Eclipse installation downloaded…"*

   Please note that all the steps describe changes to be done in your Eclipse installation, **not in the target platform**.
5. Install Eclipse Subversive plugin for managing source code from a SVN repository. In order to do this, please go to http://www.eclipse.org/subversive/downloads.php page and follow the instructions provided in the "Install the Latest Stable Build" section there.
   - Select the option presented in the screenshot below

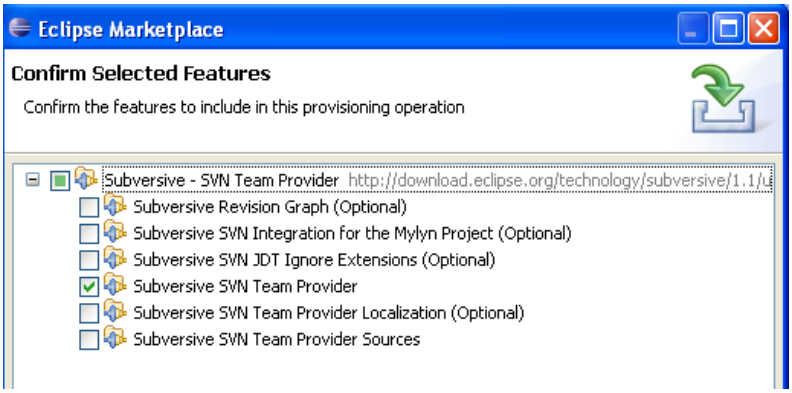


Figure 17. SVN option presented in the screenshot

- Eclipse IDE will be restarted. Select a newly installed perspective: “SVN Repository Exploring”

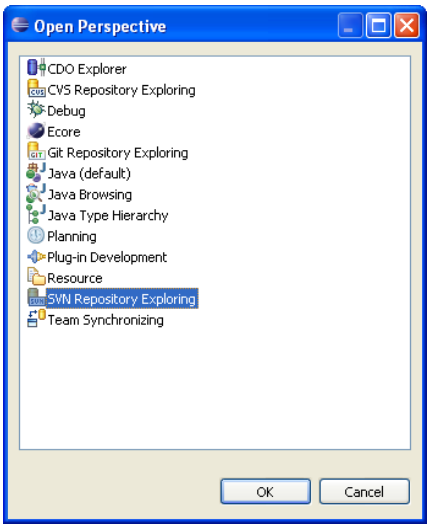


Figure 18:Opening SVN Repository Explorer

- “Install SVN Connectors” window will appear. Mark “SVN Kit 1.7.11” and press Finish

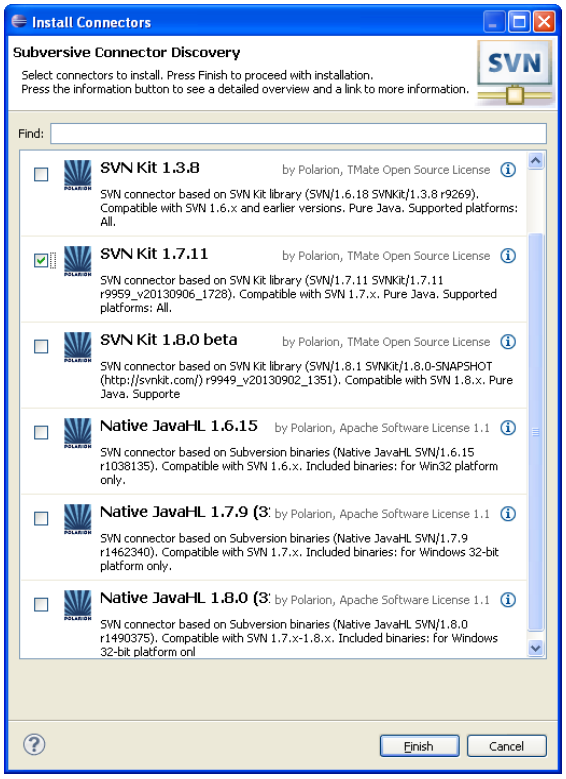


Figure 19.SVN connectors

- A new window will appear. Select both options showed on the dialog.

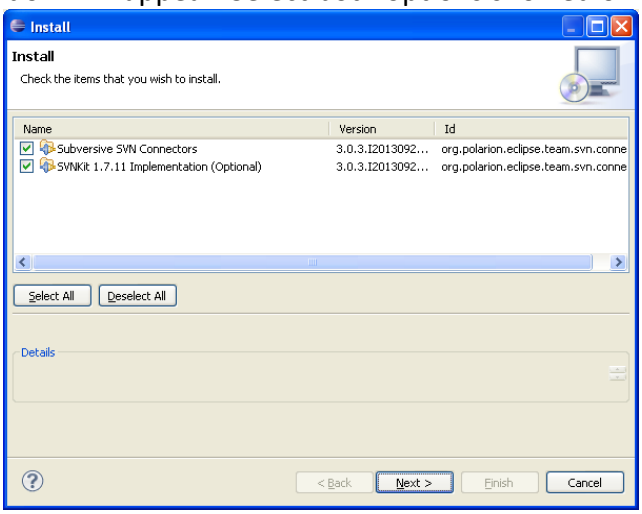


Figure 20. Installing connector

- The Subversive plugin should be fully installed now.

4.1.2.3 Check out OPENCOSS platform server source code

- In your Eclipse IDE, please go to SVN Perspective. Press "*Add the new repository*" button and provide the OPENCOSS source code repository URL https://svn.win.tue.nl/repos/opencoss-code, and a valid SVN user and password.

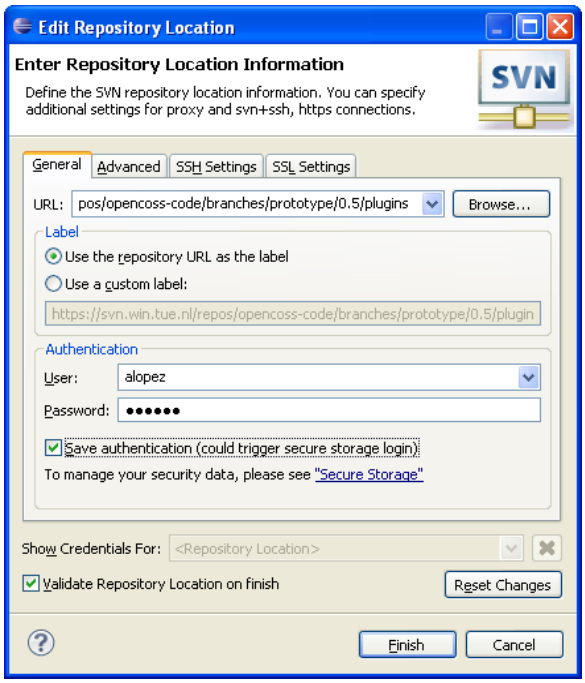


Figure 21. Entering repository connection

- Please select and check out the following projects into your workspace:
  - All projects from *trunk/common/**

o The following projects from *trunk/prototype/plugins*:

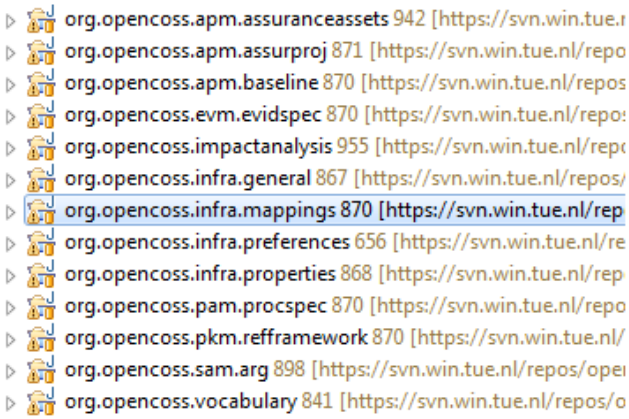


Figure 22. Project trunks

- All the projects source code should compile.

4.1.3 Running the server in Eclipse debugger

4.1.3.1 Setting up Apache Tomcat webserver

1. Download Apache Tomcat from http://tomcat.apache.org/download-70.cgi. Unpack it in your target folder.

2. In Eclipse, define *tomcat_home* variable by choosing: *Run -> Debug configurations -> Java Application -> org.opencoss.webapps -> Arguments*

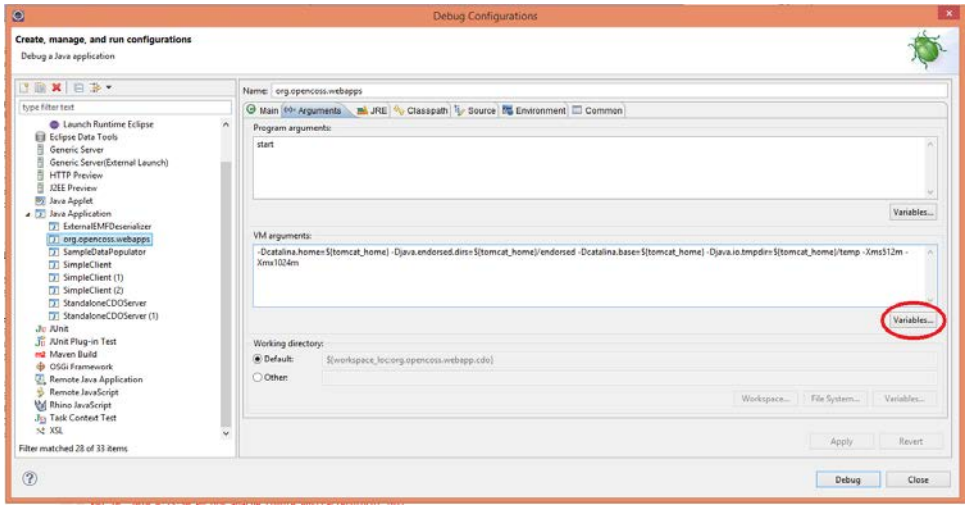


Figure 23. Configuring Eclipse TOMCAT

Edit Variables -> New
Please enter the following data:
Name – tomcat_home,

Value – [path to your Apache Tomcat folder]. This value will be referred to as [TOMCAT_FOLDER] in the remaining part of this document.

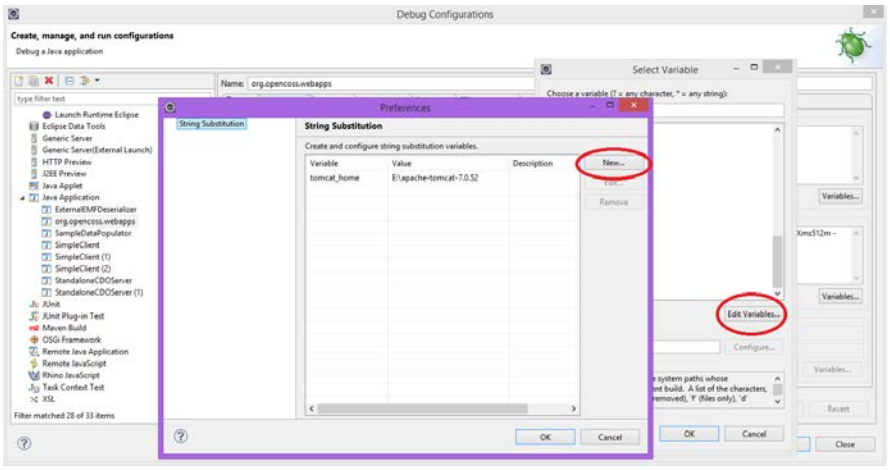


Figure 24. Configuring TOMCAT folder

3. Override your *[TOMCAT_FOLDER]\conf\server.xml* file with *org.opencoss.build\tomcat\conf\server.xml* file from your workspace.

4. Edit this file and modify , *docBase* and *workDir* attributes of *<context>* elements.
   For *docbase* – please enter a path to *webapp* subfolder of your respective project location in workspace.
   For *workDir* – please enter a path to *work* subfolder of your respective project location in workspace.
   For example:

For *cdo:*

```
<Context path="cdo" reloadable="true"
docBase="/home/dariuszo/workplace/code-staging/org.opencoss.webapp.cdo/webapp"
workDir="/home/dariuszo/workplace/code-staging/org.opencoss.webapp.cdo/work”
 …
</Context>
```

After modifications:

```
<Context path="cdo" reloadable="true" docBase="
d:\home\john\OPENCOSS_WORKSPACE/org.opencoss.webapp.cdo/webapp"
workDir="d:\home\john\OPENCOSS_WORKSPACE/org.opencoss.webapp.cdo/work”
 …
</Context>
```

For *opencoss-report:*

```
<Context path="opencoss-report" reloadable="true"
docBase="/home/dariuszo/workplace/code-staging/org.opencoss.webapp.reports/webapp"
workDir="/home/dariuszo/workplace/code-staging/org.opencoss.webapp.reports/work” …
</Context>
```

After modifications:

```
<Context path="opencoss-report" reloadable="true"
docBase="d:\home\john\OPENCOSS_WORKSPACE/org.opencoss.webapp.reports/webapp"
workDir="d:\home\john\OPENCOSS_WORKSPACE/org.opencoss.webapp.reports/work" … </Context>
```

5. Define Apache Tomcat user by adding the below section to *<tomcat-users>* node in *[TOMCAT_FOLDER]\conf\tomcat-users.xml* file:

```
<tomcat-users>
        <role rolename="manager-gui"/>
        <user username="tomcat" password="tomcat" roles="manager-gui"/>
</tomcat-users>
```

User and password to your local Tomcat will be: tomcat/tomcat.

6. Copy *DevloaderTomcat7.jar* from *org.opencoss.build\devloader* to *[TOMCAT_FOLDER]\lib*.

4.1.3.2 Setup a workspace to run a debugger

1. Below we use term “system user home directory”. Depending on your system this might be:
   - Windows XP: *c:\Documents and Settings\<username>*
   - Windows 7/8: *c:\Users\<username>*
   - Linux: */home/<username>/*

2. Create *devloaderWorkspaces* file in your system user home directory and fill it with information about location of “workspace1” (being the OPENCOSS eclipse workspace root). This file content should look similar to this:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE properties SYSTEM "http://java.sun.com/dtd/properties.dtd">
<properties>
        <entry key="workspace1">/home/dariuszo/workplace/eclipse.opencoss.workspace</entry>
</properties>
```

3. Adjust OPENCOSS server configuration file settings.

   Go to *org.opencoss.build/conf-opencoss* folder and move *opencoss-properties.xml* file from this location to the operating system user home directory. This is the location from where OPENCOSS server reads *opencoss-properties.xml* settings file.

   Edit *opencoss-properties.xml* settings file.
   The most important entries in this file are:

- *"dbUser" / "dbPassword"*
  These are PostgreSQL user credentials. Please specify a valid user and password for your PostgreSQL server.

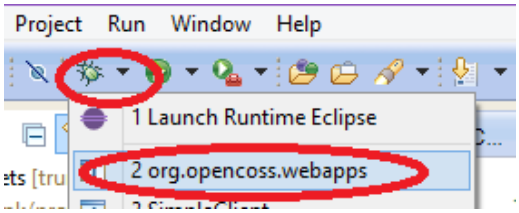
- *"serverAddress"*
  This is CDO repository name which is broadcasted by the CDO server. The "localhost" default value should be replaced with the specific server machine host name in order OPENCOSS tool clients are able to connect to this server repository from other hosts. Please modify the following entry:

  *<entry key="serverAddress">**localhost**:2036</entry>*

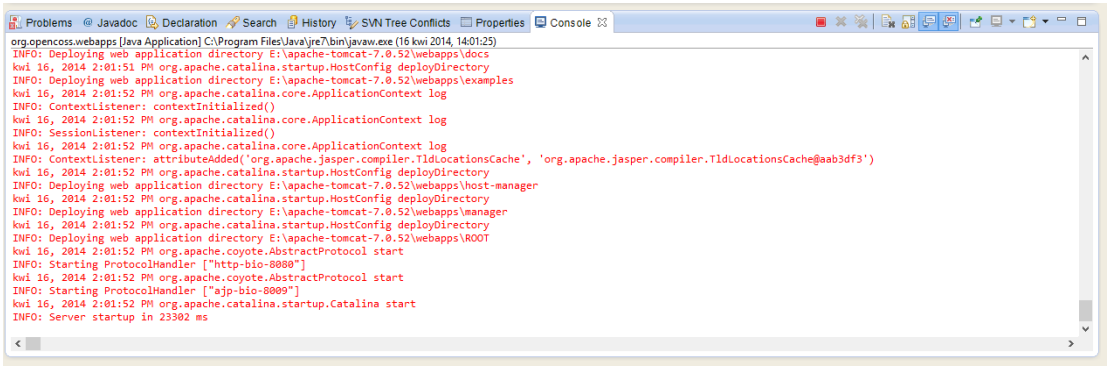
  by replacing "localhost" with the specific server host name, e.g.:

  *<entry key="serverAddress">**host-name.acme.com**:2036</entry>*

4. Run project in debug mode:



5. The following messages should be displayed on the console:



6. Run your web browser and enter http://localhost:8080 - Apache Tomcat home page should be presented.
   OPENCOSS server web reports should be accessible at http://localhost:8080/opencoss-reports/ .

4.1.4 Building OPENCOSS server web application war files

OPENCOSS server source code comes with an automation script that supports the building of web application war files from the code projects. The script has been developed in Gradle technology. A configuration procedure and build scripts execution are described below.

4.1.4.1 Installation of Gradle framework

- Download Gradle bundle from http://www.gradle.org/installation
- Unzip the downloaded Gradle package to the target folder, e.g. "C:\Program Files". A *gradle-x.x* subdirectory will be created from the archive, where x.x is the version number.
- Add the location of your Gradle "bin" folder to your operating system PATH variable. For example on Windows this can be done by opening system properties window (WinKey + Pause), selecting "Advanced system settings" tab, pressing "Environment Variables" button, and then adding the bin folder path (e.g. *"C:\Program Files\gradle-x.x\bin"*) to the end of your PATH variable. Please make sure not to use any quotation marks for the path value even if it contains spaces.
- In the same dialog, make sure that JAVA_HOME exists in your user variables or in the system variables, and it is set to the location of your JDK (e.g. C:\Program Files\Java\jdk1.7.0_06) and that %JAVA_HOME%\bin is in your PATH environment variable.
- Open a new command prompt (on Windows type *cmd* in Start menu) and run
  gradle –version
  to verify that the framework has been installed correctly.

4.1.4.2 Configuration of environment

- Go to org.opencoss.build/gradleCopyToWorkspaceRoot folder and copy *build.gradle* and *settings.gradle* files from this location to your Eclipse workspace.
- Edit *build.gradle* file and adjust tomcatHome variable to your local Apache Tomcat location:

```
tomcatHome='/home/john/workplace/tools/[TOMCAT_FOLDER]'
```

4.1.4.3 Building web application war files

- In the command prompt, go to your workspace folder.

- Run *gradle clean* command.

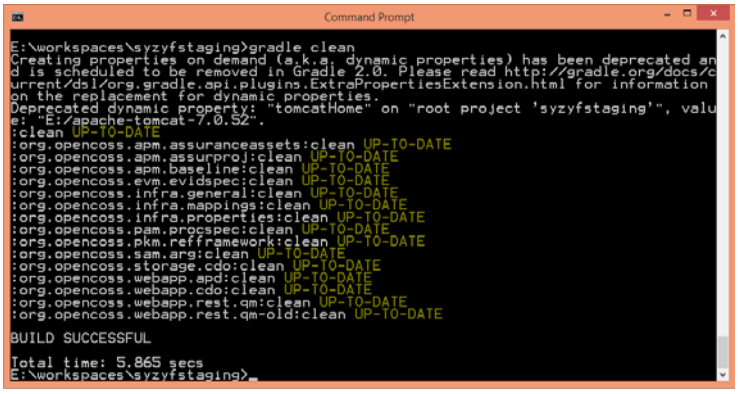


Figure 25. Command line : Run *gradle clean* command

- Run *gradle* command.

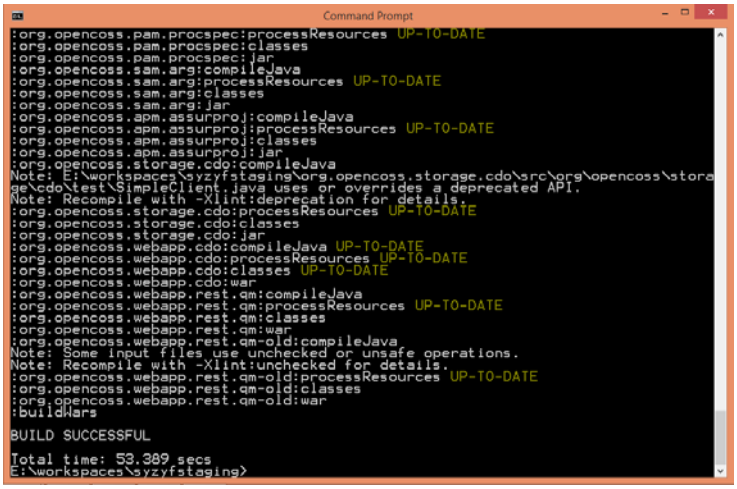


Figure 26 Command line : Run gradle command.

- The web application war files should be created in the *\build\libs* output folders on the following projects:
  org.opencoss.webapp.reports
  org.opencoss.webapp.rest.qm
  org.opencoss.webapp.cdo
  org.opencoss.webapp.apd

4.2 Developer manual - Client

A document for the developers has been elaborated, which includes step by step instructions of how to set up their development environments with all the 2nd prototype's client source code. This guide can be found at:

https://svn.win.tue.nl/repos/opencoss-code/trunk/prototype/doc/OPENCOSS_Prototype2_Client_DeveloperGuide.doc

5 Research conducted and plans for future implementation

Parallel to the core software development activities for D6.6, additional research has been conducted in order to pave the way for future implementation. The main research areas are described in the following subsections.

5.1 Evidence Evaluation

Evidence evaluation is one of the functional areas defined for the evidence management service infrastructure in D6.2. This functional area is concerned with the assessment of the completeness and adequacy of the body of evidence of an assurance project, and the assessment of specific criteria defined for evaluation of individual evidence items. Functionality has been presented above for assessing the completeness and adequacy of the body evidence with regards to the baseline of an assurance project (gap analysis functionality). In addition, work has been performed for the evaluation of individual evidence items.

This work has been developed in the scope of both WP5 and WP6, as evidence evaluation can depend on the argumentation structure for which evidence is used. Details about the work can be found in WP5 deliverables. For clarity and simplicity, we have decided to include the detailed information only in WP5 instead of dividing the information among WP5-6 deliverables. We summarize the work as follows, so that the reader gets an overall understanding of it:

1. Experts on safety assurance and certification were interviewed for eliciting the information that makes them develop confidence in evidence items
2. A taxonomy of criteria was then specified based on the insights provided by the interviewees (Figure 3)
3. Afterwards, a set of generic questions and a process were specified for evaluating evidence items according to the criteria
4. Currently, tool support is under development and the approach is being evaluated with information from past safety assurance and certification projects. It is expected to integrate the tool into the overall OPENCOSS tool platform.

In essence, the approach and tool developed for evaluation of individual evidence items can be used for the items in isolation (mainly WP6 scope) or in the context of an argumentation structure (mainly WP5 scope).

5.2 Impact Analysis

Impact analysis, and more concretely evidence change impact analysis, is another functional area defined for the evidence management service infrastructure (see D6.2). It is concerned with the identification and analysis of possible effects resulting from changes in the body of evidence of an assurance project. We have presented above the impact analysis functionality that has already been implemented. In addition to this, work on impact analysis has been performed for analyzing in depth current practices and needs in industry and determining specific impact analysis areas that might require our attention.

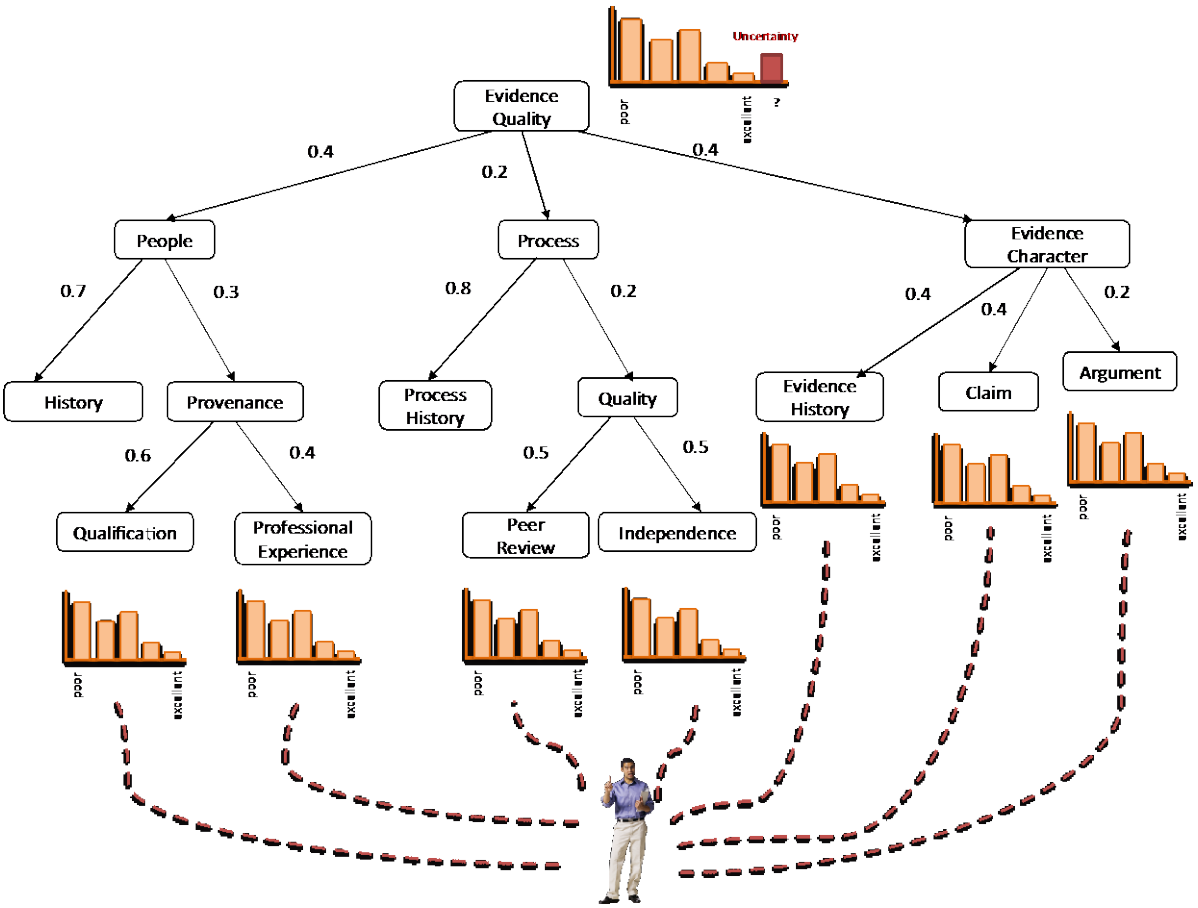


Figure 27. **Overview of the approach for evaluation of individual evidence items**

We introduced a questionnaire-based survey on evidence change impact analysis in D6.3. The questions referred to the circumstances under which safety evidence change impact analysis is addressed, the tool support used, and the challenges faced. We obtained 97 valid responses representing 16 application domains (including automotive, avionics, and railway), 28 countries, and 47 safety standards. The results suggest that safety evidence change impact analysis is most frequently addressed during system development and mainly based on system specifications changes, and that the level of automation is low. In fact, the most common challenge is insufficient tool support. Some outstanding findings are that over 40% of the respondents have dealt with system re-certification for a different application domain and 50% for a different standard, safety case evolution should be better managed, and no commercial impact analysis tool has been reported as used for all artefact types. This makes us confident in the importance of the impact analysis work in OPENCOSS, including (re)certification across product versions, systems, standards, and domains, and in the need for its results. Guidance based on the analysis of the results will be defined in D6.7 (Evidence management service infrastructure: Methodological guidance).

The impact analysis areas that might be further studied include:

1. Types of relationships between artefacts/pieces of evidence (e.g., structural) and implications in the type of change impact

2. Types of change impact between specific types of artefacts (e.g., requirements and source code) and their consequences
3. Implications for impact analysis of the evolution and (re)use of artefacts/pieces of evidence (different versions of an artefact, component reuse in several projects...)
4. Artefact change impact on other assurance assets (activities, arguments, claims…), and vice-versa
5. Impact analysis as a means for facilitating decision making (e.g., regarding cost and time) in situations such as component reuse and (re)certification against a different standard or in a different domain.
6. Use of assurance cases as the basis for impact analysis

Finally, an evidence lifecycle can be specified as a way to verify that suitable impact analysis support is provided. Available evidence lifecycles (e.g., in SACM) do not take impact analysis needs into account, and it is not possible to explicitly specify currently with the CCL if an artefact (or an assurance asset in general) is valid or invalid after a change. It has to be studied if extending the CCL is really necessary, as it seems that such information can be derived based on its current structured. In essence, impact analysis and evidence evolution with the CCL must be further studied to determine its suitability.

Future research on impact analysis is also beyond the scope of WP6. It relates to the rest of technical WPs (e.g., vocabulary aspects in WP4, argumentation aspects in WP5, and activities aspects in WP7).

5.3 Approach to OSLC

As outlined in D6.5, it has been agreed "*that OSLC is not going to be used as an implementation technology in the next prototype phase*" mainly due to the complexity of tool setup and of the required infrastructure. Furthermore, the key innovations of the OPENCOSS project are largely independent of a particular protocol or technology to exchange data between tools; the project prefers to focus on core functionalities rather than new technology adoption.

However, as mentioned in D6.5 already as well, OSLC is more than just another implementation technology but rather a complete tool integration approach. The idea of avoiding known integration pitfalls (point-to-point integration complexity, exchange, conversion and duplication of a huge amount of data) by above all keeping the data in the respective domain specific tools is fully applicable to the OPENCOSS initiative. Hence integration of tools as for example QM or medini did follow this approach in general.

Secondly, OSLC describes a technology foundation that is based on open Web standards, protocols and emerging technologies such as REST and JSON. This is very close – if not equal – to what has been used in WP6 development. Interactions between tools and the OPENCOSS platform have been all realized using a RESTful API and JSON as exchange format, similar to what is proposed by OSLC, but leaving the complex aspects related to RDF, resource handling, or OAuth handshakes out.

Last but not least, there is a set of specifications already available for specific domains, though no working group is actually working on the integration of safety evidence and assurance tools. The closest match to "evidence management" in OPENCOSS is the specification done by the OSLC

"Asset Management" working group. A rough comparison was made with the general conclusion that although there are differences in the detail, the general concepts and structure are similar.

5.3.1 Comparison of OSLC Assets with CCL Artefacts

The OSLC Asset Management 2.0 Specification has been designed for "tools […] that perform asset searching and retrieval activities". An Artifact in that specifications is treated as "interchangeably with 'work product' or 'file'" while an Asset aka Asset Resource "may have zero or more artifacts."

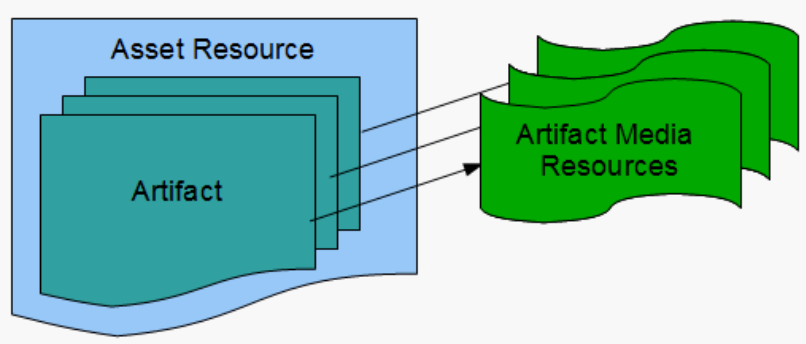


Figure 28. Simplified overview on Asset Management objects

In CCL, the metamodel looks a bit different, but not significantly. Artefacts – according to CCL spec an "individual, and identifiable unit of data managed in an assurance project", and not necessarily a file or similar kind of data – are organized in Artefact Models – similar to Asset Resource. An additional Artefact Definition acts as a logical container for the various versions of the Artefact itself. Media Resources are simply Resources with format and arbitrary location. Therefore, in principle the CCL metamodel is rich enough to hold the same kind of information as the Asset Management specification foresees for arbitrary Asset management tools. The biggest difference is that the CCL for a good reason does not foresee to really hold the resource data since that is outside the scope of OPENCOSS but rather left to external version control system and asset management tools.

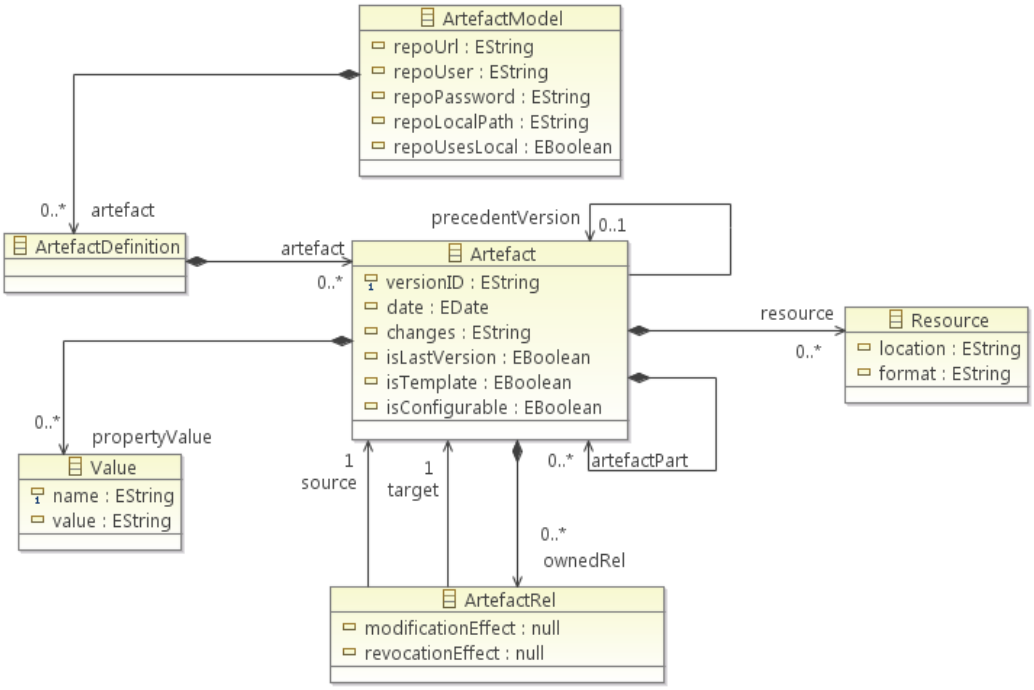


Figure 29. CCL meta model for evidence data

5.3.2 Import artifacts from tools

This use case follows the OSLC approach: evidence data are kept in the tool, simply meta-data together with full qualified references back to the imported resource are imported. Data are accessed via a REST API that is offered by the tool. A "read-only" API that only serves GET requests is sufficient here. This approach is in line with the Subversion integration of the 1st prototype.

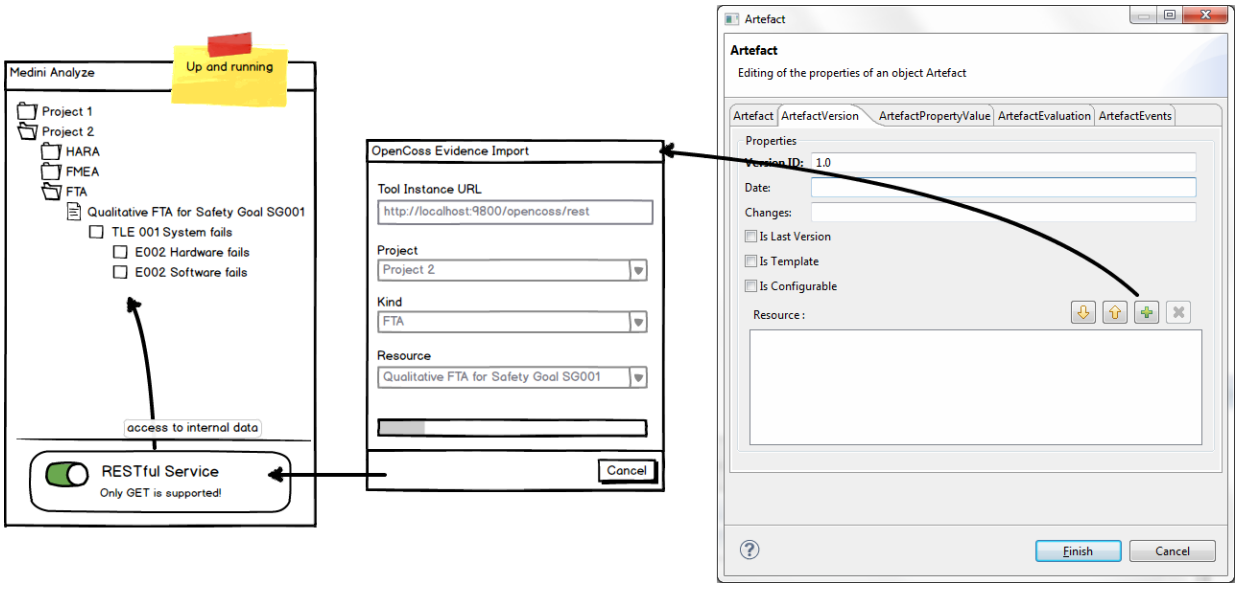


Figure 30. Mockup of a REST based pull function for artefacts from external tools

5.3.3 Export artifacts to the OPENCOSS platform

This approach is complementary to the OSLC approach since data are actively exported from the tool to the OPENCOSS platform. However, from a technological viewpoint the same techniques are used. Meta-data are exported to the OPENCOSS platform using a REST API that accepts PU or PUSH requests together with the respective JSON documents. The exported data again include references back to the original resource in the external tool so that the OPENCOSS platform can give access later and so that the meta-data are correctly traced back to the evidence data.

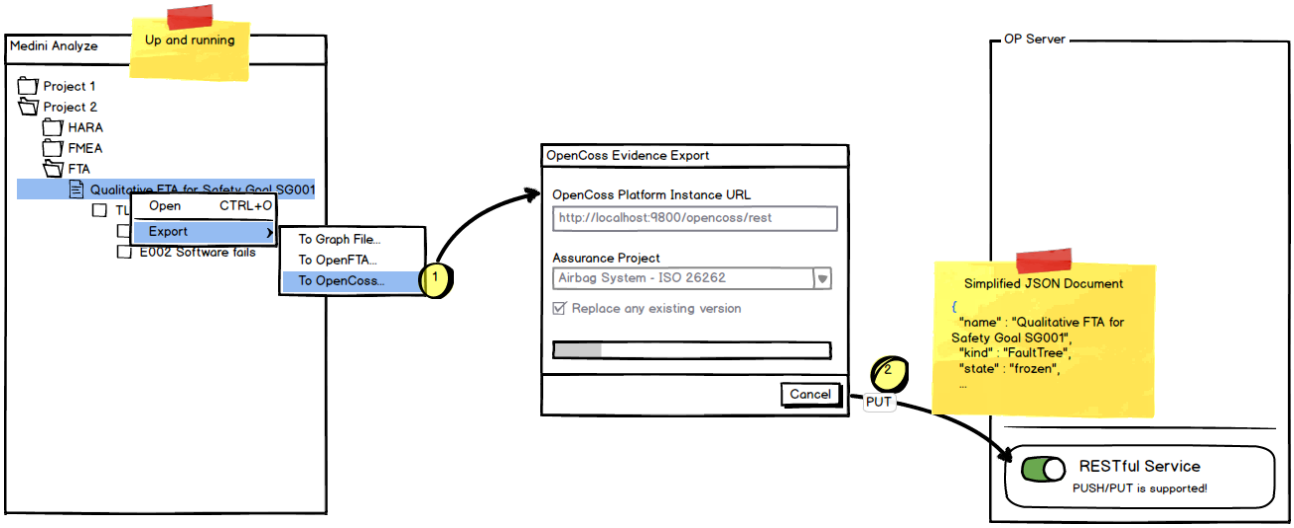


Figure 31. Mockup of a REST based push function for artefacts from external tools

5.3.4 Conclusion

It can be concluded that the integration of the OPENCOSS platform and external safety assurance tools in WP6 has followed the OSLC principles without actually being OSLC compliant. The compliance was not a criteria for the 3rd prototype phase but the general orientation towards the OSLC idea was. Besides, OSLC is heavily evolving and its success as a real de-facto standard (though likely) is still questionable. A simple mapping between the OSLC “Asset Management” specification and the CCL “Evidence Management” part seems to be feasible, thus opening the platform or the integrated tools to any OSLC consumer is a straightforward task that could be done as a follow-up task. That means that the implementation adopted by OPENCOSS does not preclude to conform to OSLC in the future.

6 Conclusion

This document has summarized and presented the details of the software development performed to implement evidence management in OPENCOSS tools done for the 2nd and 3rd prototype of the platform.

Main functional areas and the most important requirements defined in D6.2 have been implemented successfully. The less important requirements has been omitted in the final OPENCOSS prototype implementation. The detailed list of implemented requirements is available in the D3.3 document.

This document presented the exact implementation architecture, software technologies used and their configuration. It described the most important pieces of the developed code and referred to specific source code in the repository. It also presented the detailed user guide. Additionally, a software development manual with instructions for computer engineers has been provided.