



Adeptness

ADEPTNESS – Design-Operation Continuum Methods for Testing and Deployment under Unforeseen Conditions for Cyber-Physical Systems of Systems

EUROPEAN COMMISSION
Horizon 2020
H2020-ICT-01-2019
GA No. 871319



Deliverable No.	ADEPTNESS D4.5	
Deliverable Title	Report on traceability mechanism from operational data to development lifecycle	
Deliverable Date	2022-03-31	
Deliverable Type	Report	
Dissemination level	Public	
Written by	UES	2021-10-31
Checked by	MGEP, EGM	2022-04-06
Approved by	Executive board	
Status	V1.4	2022-04-26

H2020-ICT-01-2019 – 871319 – ADEPTNESS: Design-Operation Continuum Methods for Testing and Deployment under Unforeseen Conditions for Cyber-Physical Systems of Systems

Acknowledgement

The author(s) would like to thank the partners involved with the project for their valuable comments on previous drafts and for performing the review.

Project partners

- 1 – MGEP – Mondragon Goi Eskola Politeknikoa – ES
- 2 – ORO – Orona S. Coop – ES
- 3 – UES – Ulma Embedded Solutions S. Coop – ES
- 4 – SRL – Simula Research Laboratory S. Coop – NO
- 5 – BT – Bombardier Transportation Sweden – SE
- 6 – IKL – Ikerlan S. Coop – ES
- 7 – EGM – Easy Global Market SAS – FR
- 8 – MDH – Maelardalens Hoegskola – SE
- 9 – TUW – Technische Universitaet Wien – AT

Disclaimer:

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 871319.



Document Information

Additional author(s) and contributing partners

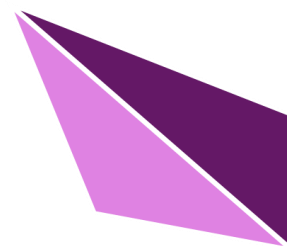
Name	Organisation
Sergio Rodríguez	UES

Document Change Log

Name	Date	Comments
V0.1	2021-10-31	Initial draft
V1.0	2022-03-09	Submitted to review
V1.1	2022-03-29	Restructured
V1.2	2022-04-07	MGEP, EGM reviews
V1.3	2022-04-11	MGEP, EGM reviews
V1.4	2022-04-26	MGEP review

Exploitable results

Exploitable results	Organisation(s) that can exploit the result
OSLC Bridge Tool	UES

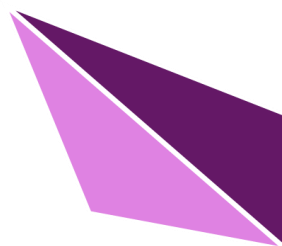


Contents

1	PURPOSE OF THE DOCUMENT	8
1.1	DOCUMENT STRUCTURE	8
1.2	DEVIATIONS FROM THE ORIGINAL DESCRIPTION IN THE GRANT AGREEMENT ANNEX 1 PART A	8
1.2.1	<i>Description of work related to deliverable in GA Annex 1 – Part A</i>	8
1.2.2	<i>Time deviations from original planning in GA Annex 1 – Part A</i>	8
1.2.3	<i>Context deviations from the original plan in GA Annex 1 – Part A</i>	8
2	INTRODUCTION	9
3	OVERVIEW	10
4	BACKGROUND AND MOTIVATION	11
4.1	BACKGROUND	11
4.2	MOTIVATION	11
4.3	OSLC (OPEN SERVICES FOR LIFECYCLE COLLABORATION)	11
4.3.1	<i>Definition</i>	11
4.3.2	<i>Organization</i>	11
4.3.3	<i>Open specifications</i>	11
4.3.4	<i>Eclipse Lyo</i>	11
4.3.5	<i>Technologies</i>	12
4.3.6	<i>RDF (Resource Description Framework)</i>	12
4.3.7	<i>Traceability</i>	12
4.3.8	<i>Domains</i>	13
5	TOOLCHAIN	16
5.1	DOCKER	17
5.1.1	<i>Overview</i>	17
5.1.2	<i>Usage in OSLC Bridge</i>	17
5.2	JAZZ TECHNOLOGY PLATFORM	17
5.2.1	<i>Overview</i>	17
5.2.2	<i>Platform</i>	17
5.2.3	<i>Usage in OSLC Bridge</i>	17
5.3	RTC (RATIONAL TEAM CONCERT™)	19
5.3.1	<i>Overview</i>	19
5.3.2	<i>Collaboration and integration across the development lifecycle</i>	20
5.3.3	<i>Process configuration and customization</i>	20
5.3.4	<i>RTC Interfaces</i>	20
5.3.5	<i>Usage in OSLC Bridge</i>	20
5.4	STELLIO / NGSI-LD	21
5.4.1	<i>Overview</i>	21
5.4.2	<i>NGSI-LD Data Model</i>	21
5.4.3	<i>NGSI-LD Mapping in Stellio</i>	22
5.4.4	<i>Usage in OSLC Bridge</i>	22

5.5	MAVEN.....	23
5.5.1	Overview.....	23
5.5.2	Usage in OSLC Bridge	23
5.6	OPENAPI	24
5.6.1	Overview.....	24
5.6.2	YAML.....	24
5.6.3	Swagger.....	24
5.6.4	Swagger UI framework.....	24
5.6.5	OpenAPI Generator (Swagger Codegen)	24
5.6.6	Usage in OSLC Bridge	25
5.7	ECLIPSE LYO.....	30
5.7.1	Overview.....	30
5.7.2	Lyo SDK.....	30
5.7.3	Usage in OSLC Bridge	30
5.8	JERSEY.....	31
5.8.1	Overview.....	31
5.8.2	Usage in OSLC Bridge	31
5.9	SPRING BOOT	31
5.9.1	Overview.....	31
5.9.2	Spring framework.....	31
5.9.3	Spring Boot solution	31
5.9.4	Usage in OSLC Bridge	31
5.10	JUNIT	31
5.10.1	Overview.....	31
5.10.2	Usage in OSLC Bridge	32
5.11	SONARQUBE.....	32
5.11.1	Overview.....	32
5.11.2	Usage in OSLC Bridge	32
6	OSLC BRIDGE.....	33
6.1	OVERVIEW.....	33
6.2	ENVIRONMENTS.....	34
6.2.1	Local Development Environment.....	34
6.2.2	Final environment.....	34
6.3	ARTEFACTS.....	35
6.4	ARCHITECTURE.....	36
6.4.1	Spring Boot (Spring.io).....	36
6.4.2	Interface.....	36
6.4.3	OSLC Bridge Business	37
6.5	VALIDATION MODEL	40
6.5.1	Overview.....	40
6.5.2	Entities.....	40
6.6	OPERATION.....	42
6.6.1	Setup	42

6.6.2	<i>Notification</i>	43
6.6.3	<i>Traceability</i>	45
6.7	BUILD.....	46
6.8	DEPLOY.....	47
7	SUMMARY	50
8	RISK REGISTER	51
9	REFERENCES	52
10	QUALITY ASSURANCE	53
11	ACKNOWLEDGEMENTS	54

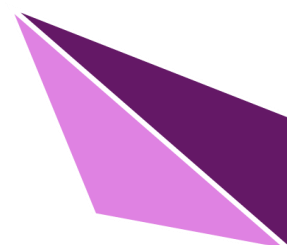


List of figures

Illustration 1. OSLC Bridge connections.....	10
Illustration 2. Linked Lifecycle Data OSLC.....	12
Illustration 3. OSLC Requirements Management	14
Illustration 4. OSLC Change Management Overview.....	14
Illustration 5. OSLC RM Domain Resources.....	15
Illustration 6. Toolchain.....	16
Illustration 7. Development tools and environments.	16
Illustration 8. Jazz Team Server login.	19
Illustration 9. Rational Team Concert flow [6].....	20
Illustration 10. Local development environment.....	34
Illustration 11. Final environment	35
Illustration 12. OSLC Bridge architecture.....	36
Illustration 13. Validation model.....	40
Illustration 14. Testcase with verdict as parameter.....	41
Illustration 15. Testcase based model	45
Illustration 16. OSLC Bridge running from CLI.	46
Illustration 17. OSLC Bridge running from IDE.....	47

List of tables

Table 1. Link types.....	13
Table 2. OSLC Domains.....	13
Table 3. NGSI-LD entity	21
Table 4. NGSI-LD property	21
Table 5. NGSI-LD relationship.....	22
Table 6. Stellio NGSI-LD Context broker.....	23
Table 7. Adeptness OSLC Bridge methods.	30
Table 8. OSLC Bridge dependencies	30
Table 9. OSLC Bridge tests.....	32
Table 10. Ubuntu server	34
Table 11. Developed artefacts.....	36
Table 12. RTCUtils methods	38
Table 13. OSLC Bridge Factories.....	38
Table 14. StellioUtils methods.....	39
Table 15. followRelations method.....	39
Table 16. Mapping Adeptness model with OSLC Domain.....	41
Table 17. Mapping Adeptness model with OSLC Domain.....	41
Table 18. Setup RTC.....	42
Table 19. Setup Stellio.	42
Table 20. OSLC attributes.....	45



1 PURPOSE OF THE DOCUMENT

The purpose of this document is define the traceability mechanism from operational data to development lifecycle using the developed OSLC Bridge.

1.1 Document structure

- Section 2 presents the Introduction.
- Section 3 provides the Overview.
- Section 4 presents an introduction to the background and motivation along with the OSLC description.
- Section 5 explains the toolchain involved in the development and exploitation of the OSLC Bridge.
- Section 6 presents the OSLC Bridge along with different environments, developed artefacts and the OSLC Bridge's flow.
- Section 7 provides the summary
- Section 8 provides the risk register.
- Section 9 presents the references.
- Section 10 presents the quality assurance.
- Section 11 provides the acknowledgement.

1.2 Deviations from the original Description in the Grant Agreement Annex 1 Part A

1.2.1 *Description of work related to deliverable in GA Annex 1 – Part A*

There are no deviations with respect to the work of this deliverable.

1.2.2 *Time deviations from original planning in GA Annex 1 – Part A*

Deliverable date was initially delayed from M24 (2021-12-31) to M27 (2022-03-31) by means of an amendment. Moreover, due to major revision request, four additional week delay has been added (including two week Easter holidays period).

1.2.3 *Context deviations from the original plan in GA Annex 1 – Part A*

There are no deviations from Annex 1.



2 INTRODUCTION

In the operation of CPSoS, unexpected events and unforeseen situations can happen. In this context, it is important to know which lifecycle artefacts are affected by situations that may occur at develop or operation time to reduce the time spent in impact and traceability analysis.

This task aims to focus on the traceability of the operational part with lifecycle artefacts. This enables tracing operational data given during a determined time window with lifecycle artefact by employing the OSLC standard to this end.

More information about OSLC in the section [4.3 OSLC \(Open Services for Lifecycle Collaboration\)](#).



3 OVERVIEW

The traceability in the context of the Adeptness project is realized through the OSLC Bridge. The OSLC Bridge is an Adeptness-specific implementation bringing together OSLC world and Adeptness NGSI-LD defined entities, microservices, attributes and resources.

More information about OSLC and NGSI-LD tools in D5.2_Toolchain deliverable.

The OSLC Bridge is a connector between OSLC (Open Services for Lifecycle Collaboration) and Stellio Context Broker.

- OSLC defines a set of specifications that enable integration of software development and provides the OSLC providers access to RM (Requirements Management), AM (Architecture Management), CCM (Configuration Management) and QM (Quality Management) domains. More information of those domains in section [4.3.8 Domains](#).
- Stellio is a context broker that implements a NGSI-LD specification. This context broker contains the definition of Adeptness Validation Model.

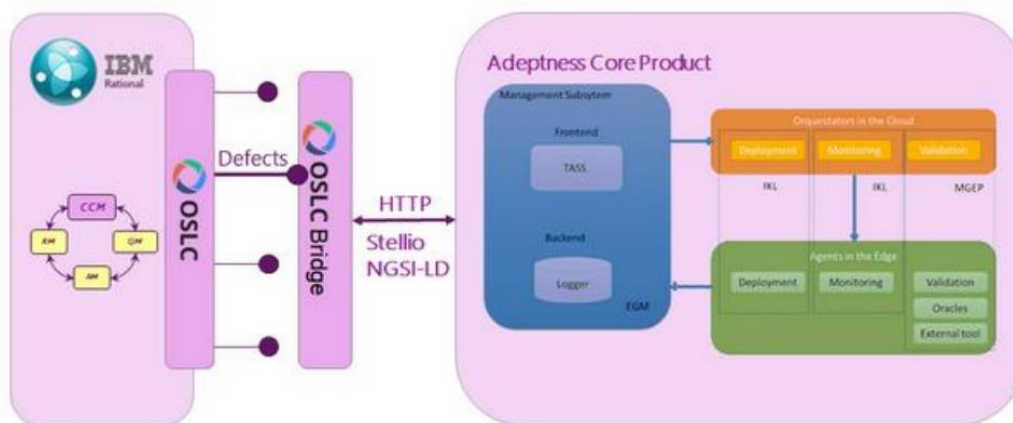


Illustration 1. OSLC Bridge connections.

The overall ecosystem does not only manage the CPSoS seamless interoperability, but also enables lifecycle collaboration, from requirements management, and design, to verification and validation artefacts. This collaboration provides a complete picture of the product lifecycle management and application lifecycle management, easing the traceability from conception to runtime execution.



4 BACKGROUND AND MOTIVATION

4.1 Background

Standardised OSLC APIs often allow vendors to provide a fully supported integration with many other OSLC compliant out of the box tools.

An OSLC integration can be performed not only at the level of tool data models but also at the level of the workflow involving those tools.

This deliverable presents an OSLC bridge for collecting validation results and triggering issues by means of OSLC standard.

4.2 Motivation

The OSLC standard selection has been motivated by the experience that ULMA Embedded Solutions has in the management and development of projects under the OSLC standard, specifically under the Jazz Team Server tool.

4.3 OSLC (Open Services for Lifecycle Collaboration)

4.3.1 Definition

OSLC [\[1\]](#) is a set of defined specifications that allow the integration of software development. OSLC is constantly evolving in areas such as Product Lifecycle Management (PLM), Application Lifecycle Management (ALM), IT Operations and more.

4.3.2 Organization

OSLC initiative is split up into different OASIS TC (Organization for the Advancement of Structured Information Standards Technical Committees). For a context of a specific part of the lifecycle exists an OASIS Technical Committee that defines developments specifications.

A Core technical committee defines a common specification that is extended in turn by each lifecycle technical committee.

4.3.3 Open specifications

The OSLC initiative is open to participate in the specification through OASIS Technical Committees. Participants who want to take part in the specification, have to sign the Intellectual Property Rights policies in order to ensure irrevocability.

4.3.4 Eclipse Lyo

The Eclipse Lyo [\[2\]](#) project is a Java reference implementations for OSLC standard. It is an open-source project providing consumer and provider SDKs, reference implementations, samples and test suite. Likewise, Eclipse Lyo enables the interoperability of services, products, and other distributed network resources.

The promotion of the use of Linked Data principles along with the OSLC standard for publishing lifecycle are part of the goals of the Eclipse Lyo project. The open OASIS OSLC standard is based on RESTful architecture



and Linked Data principles, such as those defined in the RDF family of specifications, and the W3C Linked Data Platform.

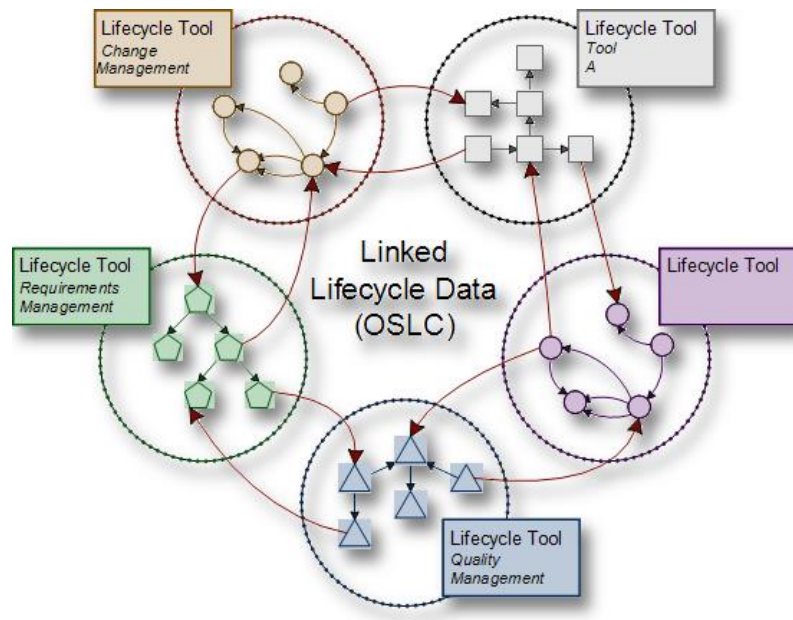


Illustration 2. Linked Lifecycle Data OSLC.

4.3.5 Technologies

These are some technologies involved:

- W3C Resource Description Framework (RDF)
- W3C Linked Data that is used to define the OSLC resources as RDF properties
- REST operations on resources that are performed using HTTP connections.

Hinging on afore mentioned technologies OSLC is capable of enabling integration at data level via links between related resources.

4.3.6 RDF (Resource Description Framework)

RDF [3] defines a standard model for data interchange on the World Wide Web. The RDF standard defines features to facilitate data merging, and it specifically supports the evolution of schemas over time without requiring all the data consumers to be changed.

4.3.7 Traceability

Aiming at incorporating the traceability mechanism from operational data to development lifecycle, the OSLC Bridge makes use of OSLC as it defines a set of specifications that enables integration of software development. In addition, OSLC allows traceability with OSLC standard link types supported as we can see in the following table.

Project relationship	Link types
Related To	Affected By Defect, Affects Plan Item and Related Change Request



Tested By	Affects Test Result, Tested By Test Case
Implements	Affects Requirement, Implements Requirement
Tracks	Related Test Case, Test Execution Record, Test Plan, Test Script and Test Suite
Tracks Requirement	Tracks Requirement

Table 1. Link types

4.3.8 Domains

OSLC comprehends a set of specifications for integrating lifecycle tools. This set allows conforming independent software and product lifecycle tools to integrate both their data and workflows in support of end-to-end lifecycle processes.

The underlying concept in OSLC is to enable traceability and interaction across domains. The specifications of OSLC for each domain provide a complete definition of such domain which are linked to the OSLC Core specification. The domains are defined in the next table.

Domain	Short name	Description
Requirements Management	RM	OSLC Requirements Management defines the OSLC services and vocabulary for the Requirements Management domain. Requirements Management define the management of Requirements, Requirements Collections and supporting resources defined in the OSLC Core specification.
Change Management	CM	OSLC Change Management defines the OSLC services and vocabulary for the Change Management domain. Change Management specifies resources for the management of change requests for the product, activities, tasks, and relations between those and related resources in other domains such as test cases, requirements, or architectural resources.
Configuration Management	CCM	OSLC Configuration Management defines a set of REST APIs for managing versions and configurations of linked data resources from multiple domains and an RDF vocabulary for Configuration Management
Architecture Management	AM	OSLC Architecture Management defines the OSLC services and vocabulary for the Architecture Management domain. Architecture Management resources address the management of product design artefacts including models and relationships with other resources such as requirements, testing resources and change requests.
Quality Management	QM	OSLC Quality Management defines the OSLC services and vocabulary for the Quality Management domain. Quality Management resources define the test cases, test plans, and test results for lifecycle of the software delivery.

Table 2. OSLC Domains



4.3.8.1 Requirements Management

Requirements Management (RM) is an OSLC specification that defines the properties, resources and operations to be supported by a server of OSLC Requirements Definition and Management. OSLC Core lasts versions are currently in public review as candidates for OASIS Standard.

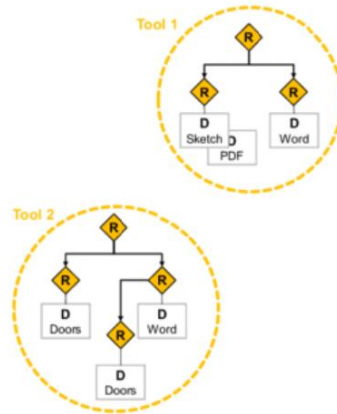


Illustration 3. OSLC Requirements Management.

4.3.8.2 Change Management

Change Management (CCM) is a RESTful web services interface that allows the management of activities, tasks and relationships related to the product change request, like test cases, requirements or architectural resources.

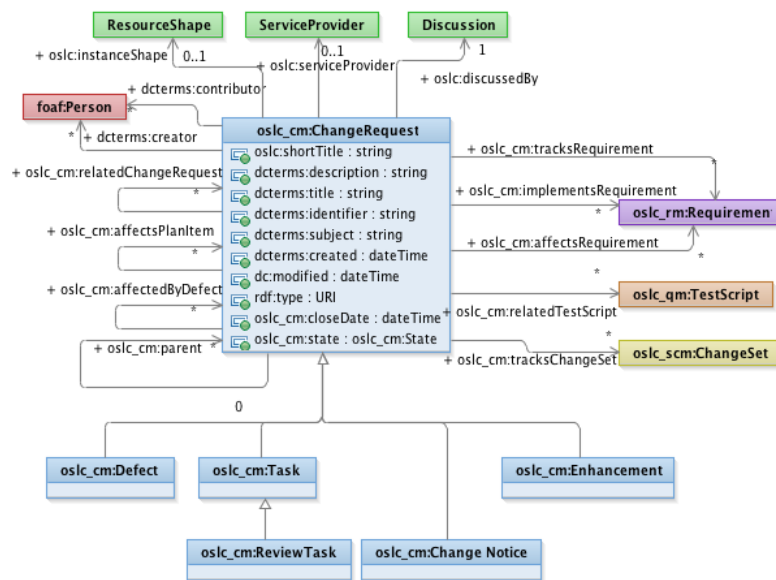


Illustration 4. OSLC Change Management Overview.

4.3.8.3 Quality Management

Quality Management (QM) defines the test plans, test cases, and test results, along with the lifecycle of the software delivery that should be implemented in an OSLC Quality Management provider.



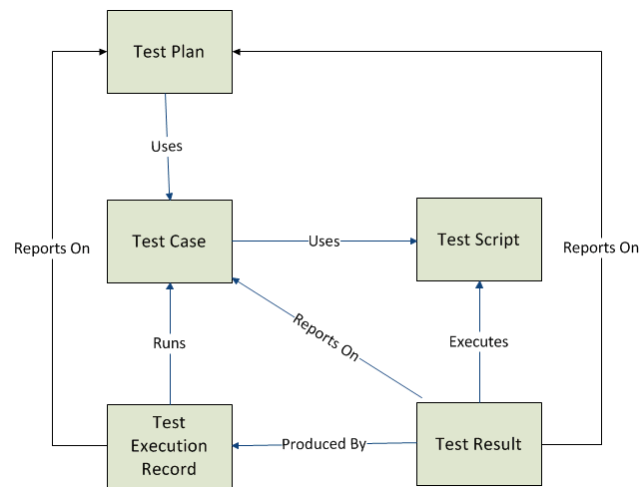


Illustration 5. OSLC RM Domain Resources.



5 TOOLCHAIN

There are several actors involved in the development of the OSLC Bridge in a **local environment** (see chapter 6.2), including the ones presented and highlighted in yellow in the following figure.

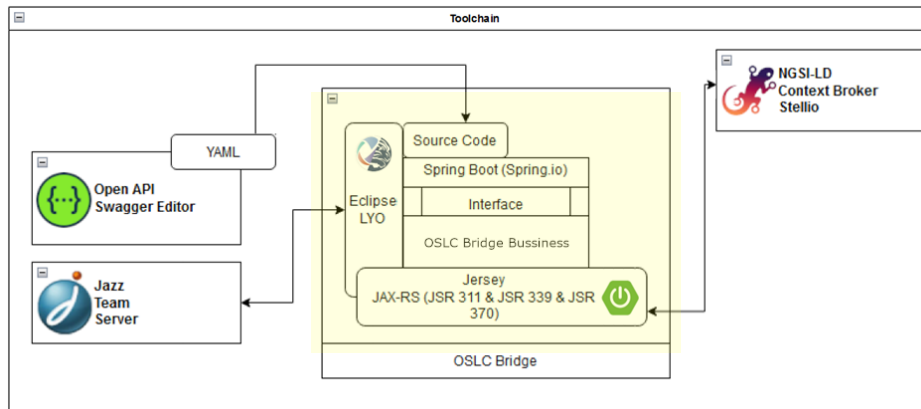


Illustration 6. Toolchain.

- The **Source Code** of the OSLC Bridge is organized in two decoupled parts connected via a simple interface.
- **Jersey** provides a REST framework that implements JAX-RS. This source code is automatically generated with OPEN API using a YAML file for API REST definition.
- **Eclipse Lyo** supports the development of REST-based servers and clients in Java managing the information as RDF resources.

The choice of the usage of Jersey as a framework is due to the possibility of sharing libraries, as Eclipse Lyo uses Jersey internally.

In the **final environment** (see chapter 6.2), tools from toolchain presented in the following sections are also used to build and deploy the OSLC Bridge.

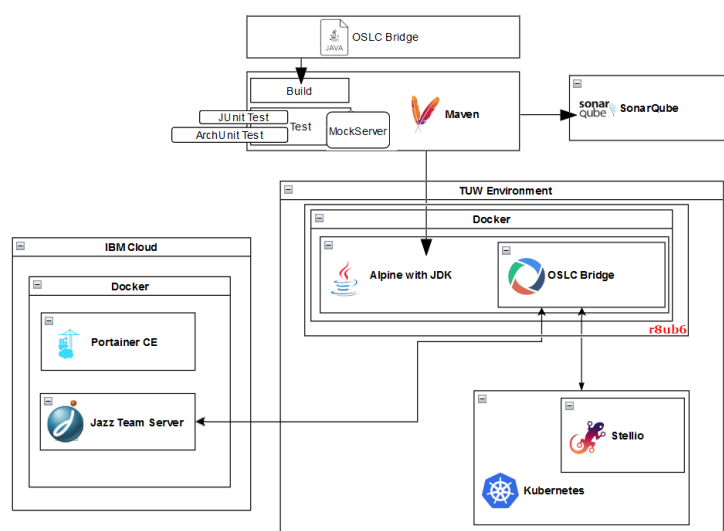


Illustration 7. Development tools and environments.



5.1 Docker

5.1.1 Overview

Docker [\[4\]](#) is a PaaS (platform as a service) solution that uses virtualization to deploy software in packages called containers. These containers are isolated from each other and bundle their own software, libraries and configuration files. They can communicate with each other through well-defined channels. The software that hosts the containers is called Docker Engine.

Since all of the containers share the services of a single operating system kernel, they use fewer resources than virtual machines. Likewise, Docker can use different interfaces to access virtualization features of the Linux kernel..

5.1.2 Usage in OSLC Bridge

Docker is used to host the infrastructure in development environments and the OSLC Bridge in the final environment in TUW.

5.2 Jazz technology platform

5.2.1 Overview

The Jazz platform integrates all the tasks across systems and the software development lifecycle. It provides useful building blocks and frameworks to ease the development of tools and products.

5.2.2 Platform

The Jazz technology platform supports distributed development teams, provides scalable solutions from small teams up to large enterprises and helps teams manage all lifecycle of systems and software development. The Jazz technology platform is developed at jazz.net.

5.2.3 Usage in OSLC Bridge

Rational Team Concert (RTC) is part of Jazz Team Server (JTS). OSLC Bridge uses RTC to generate issues.

JTS provides the foundational services that enable a group of applications to work together as a single logical server. After installing Jazz Team Server, applications such as Change and Configuration Management (CCM or RTC), Quality Management (QM) and Requirements Management (RM) will be available.

With the next Dockerfile file can be built the JTS image.

```
FROM jruethlin/clm603-rtc
MAINTAINER Sergio Rodriguez "srodriguez@ulmaembedded.com"
RUN echo "#!/bin/sh" > startup.sh \
    && echo "/opt/IBM/JazzTeamServer/server/server.startup" >> startup.sh \
    && echo "tail -f /dev/null" >> startup.sh \
    && chmod +x startup.sh
EXPOSE 9443
ENTRYPOINT [ "./startup.sh" ]
```

To build the image, run the command:



```
>$user sudo docker build -t ues/rtc603 .
```

We create the image ues/rtc603

Using the docker-compose.yml file with static IP

```
version: '2.1'
networks:
  jts603_default:
    ipam:
      config:
        - subnet: 172.33.0.0/24
services:
  rtc603:
    image: ues/rtc603
    container_name: rtc603
    restart: unless-stopped
    ports:
      - 9443:9443
    extra_hosts:
      # - "adeptness.ulmaembedded.com:127.0.0.1"
      - "adeptness.ulmaembedded.com:172.33.0.2"
    networks:
      jts603_default:
        ipv4_address: 172.33.0.2
    volumes:
      - rtc603_data:/opt/IBM/JazzTeamServer/server
  rtc603-exim4smtp:
    image: namshi/smtp
    container_name: rtc603-exim4smtp
    restart: unless-stopped
    ports:
      - "25:25"
    environment:
      # # To act as a Gmail relay
      - GMAIL_USER=adeptness.mailer
      - GMAIL_PASSWORD=*****
volumes:
  rtc603_data:
```

Command in /home/adeptness/dockers/jts603 path

```
>$user sudo docker-compose -f docker-compose.yml up -d
```

Open the 9443 port in the Linux firewall. It may not be necessary depending on the system.

```
>$user sudo ufw allow 9443
```

In this case, we are connecting to an HTTPS server without a valid certificate, so we obtain a Warning of potential security risk. By clicking on the advanced button, we accept the risk and continue.



We connect to <https://192.168.130.104:9443/jts> or <https://adeptness.ulmaembedded.com:9443/jts>

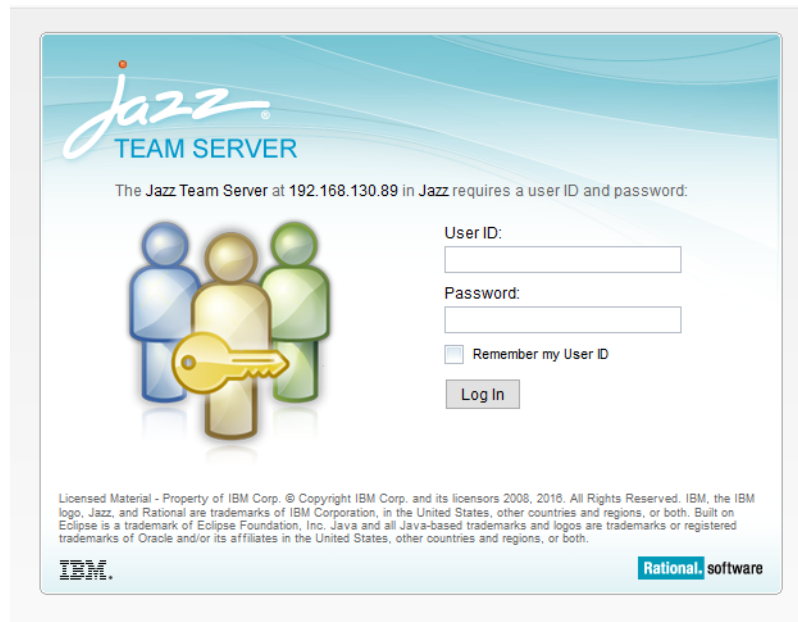


Illustration 8. Jazz Team Server login.

5.3 RTC (Rational Team Concert™)

5.3.1 Overview

Rational Team Concert™ [5] is a tool for team collaborations that integrates development tasks, like iteration planning, process definition, change management, source control, defect tracking, build automation, and reporting.

Developers use Rational Team Concert to track their work, share their changes, and collaborate. Team leads and project managers use it to plan releases and monitor progress by viewing plans, dashboards, and reports.

Rational Team Concert is the Change and Configuration Management (CCM) application in the Rational® solution for Collaborative Lifecycle Management (CLM) and the IBM® Internet of Things Continuous Engineering (IoT CE) Solution. These solutions integrate IBM Rational products to provide a complete set of applications for software or systems development. The new name for the RTC tool is **IBM Engineering Workflow Management**.



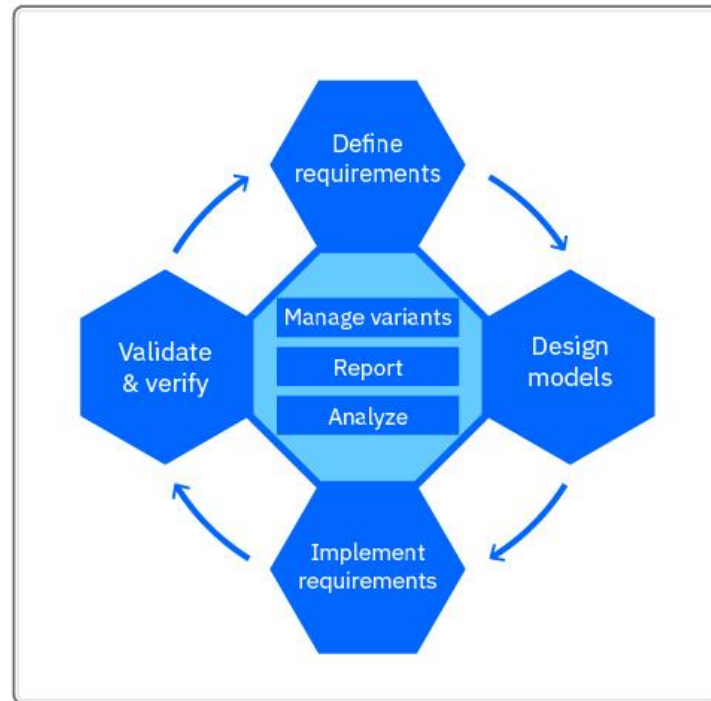


Illustration 9. Rational Team Concert flow [6].

5.3.2 Collaboration and integration across the development lifecycle

RTC exchanges information directly in the context of your work, when a work item changes, the team members are notified of the change.

With several views enabled to share team information, it can track team activity, present information in more detail, or configure which information is visible at any time.

5.3.3 Process configuration and customization

All RTC projects follow a process [6]. In RTC the collection of rules, roles, practices, permissions, and guidelines that you use to organize and control the workflow for a project is a process.

In the context of a project, a process is used to define user roles and their permissions for performing operations. Processes are defined by a process-template that can be modified and configured to enforce different rules at different points in the release.

5.3.4 RTC Interfaces

Rational Team Concert has several different interfaces, like an Eclipse client, a Microsoft Visual Studio client and a web client. These client interfaces provide the developers with an integrated and complete development environment.

5.3.5 Usage in OSLC Bridge

OSLC Bridge generate issues in Rational Team Concert from the Adeptness validation model.



5.4 Stellio / NGSI-LD

5.4.1 Overview

Stellio is an NGSI-LD compliant context broker developed by EGM.[\[7\]](#)

Three business services make up a Stellio server:

- Entity service: is a service that manages the information context. This service is backed by a neo4j database.
- Search service: handles the temporal (and geospatial) queries in Stellio. It's backed by a TimescaleDB database.
- Subscription service: manages subscriptions and notifications. It's backed by a TimescaleDB database.

In addition, two other services complete a Stellio server:

- API Gateway: is a module that dispatches requests to Stellio business services
- Kafka: is a streaming engine that decouples communication inside the broker (and allows plugging other services)

These services are developed in Kotlin and based on the Spring Boot framework and built with Gradle.

OSLC Bridge uses all the business services to search testcase entities and to subscribe to verdict values changes.

5.4.2 NGSI-LD Data Model

Stellio context broker has semantic support that allows exchanging data in the NGSI-LD format.

An NGSI-LD entity is composed of:

Property	Type	Description
Id	mandatory	The identifier of the Entity (URI format).
Type	mandatory	The type of the Entity
Properties	optional	Properties
Relationships	optional	Relationships
@context	mandatory	JSON-LD specification for linked data

Table 3. NGSI-LD entity

A property in NGSI-LD is composed of:

Parameter	Type	Description
type	mandatory	Property or GeoProperty (Stellio Semantic Support)
value	mandatory	The value
unitCode	mandatory	The unit of the value in CEFACCT format
Properties	optional	Properties
Relationships	optional	Relationships

Table 4. NGSI-LD property



A relationship in NGSI-LD is composed of:

Relationship	Type	Description
type	mandatory	Relationship (Stellio Semantic Support)
object	mandatory	The URI of the target Entity
Properties	optional	Properties
Relationships	optional	Relationships

Table 5. NGSI-LD relationship

Temporal Properties **observedAt**, **createdAt** and **modifiedAt** (Stellio Semantic Support) follow the ISO 8086 format.

5.4.3 NGSI-LD Mapping in Stellio

The proposed mapping approach for an NGSI-LD entity is as follows:

- Each entity is mapped to a subject node with the same id and type. **createdAt**, **modifiedAt** and location attributes of the NGSI-LD entity are mapped to literal properties of this node.
- Each property is mapped to an object node. The vertex relating the entity node to its property node is labelled **has_value**. The Property name, value, **createdAt**, **modifiedAt** and **unitCode** are mapped to literal properties of this node. In the case of a property of property in NGSI-LD, each property will be modelled as a node and a new **has_value** vertex relation will be created between them.
- Each relationship is mapped to an object node. **createdAt** and **modifiedAt** attribute are mapped to literal properties of this node. The vertex relating the entity node to its relationship is labelled **has_object**. As the object of a relationship in NGSI-LD is the URI of the related Entity, two vertexes labelled **has_object** and the name of the Relationship are created from the Relationship node the tagged entity node. In the case of:
 - o Relationship of Relationship: each sub-relationship is mapped to a subject node and new **has_object** vertex will link these Relationships. The sub-relationship is related to the tagged Entity via two vertexes labelled **has_object** and the name of the sub-Relationship.
 - o Property of a Relationship: each sub-property of a relationship will be mapped to a subject node and a new **has_value** vertex will be link between the relationship to its sub-property nodes.
 - o Relationship of a Property: each sub-relationship of a property is mapped to a subject node and linked via the vertex labelled **has_object**. The sub-relationship is related to the tagged Entity via two vertexes labelled **has_object** and the name of the Relationship. OpenAPI

5.4.4 Usage in OSLC Bridge

Using a docker to deploy a Stellio server in a development server or local environment.

Service	Connection	Port
Api gateway	HTTP	8080



Entity service	HTTP	8082
Search service	HTTP	8083
Subscription service	HTTP	8084

Table 6. Stellio NGSI-LD Context broker

NGSI-LD is an Open API and data model specification for context management published by ETSI.

Using the docker-compose.yml file created by EGM, then the Stellio Server is deployed.

For local development run in /home/adeptness/dockers/stellio path the command

```
>$user sudo docker-compose -f docker-compose.yml up -d
```

With this, several tools that Stellio Server needs to work, are installed and deployed.

- <https://kafka.apache.org/>
- <https://zookeeper.apache.org/>
- <https://neo4j.com/>
- <https://www.postgresql.org/>

Testing getting some ngsi-lid entities

- <http://adeptness.ulmaembedded.com:8880/ngsi-lid/v1/entities?type=DeploymentPlan>
- <http://adeptness.ulmaembedded.com:8880/ngsi-lid/v1/entities?type=DeploymentComp>
- <http://adeptness.ulmaembedded.com:8880/ngsi-lid/v1/entities?type=DeployableCompType>

The response will return an empty JSON, for being the first call, if this is a new Stellio Server recently deployed.

```
[ ]
```

5.5 Maven

5.5.1 Overview

Apache Maven is a software project management and comprehension tool. Maven is a build automation tool used primarily for Java projects.

Maven is based on a project object model (POM) concept, this POM is a central piece of information where Maven manage the project's build, reporting and documentation.

5.5.2 Usage in OSLC Bridge

Maven is used for building the OSLC Bridge.



5.6 OpenAPI

5.6.1 Overview

OpenAPI is a specification that allows RESTful web services description using machine-readable interface files. OpenAPI Specification is conformed in an OpenAPI document which may be represented in either JSON or YAML format.

An OpenAPI document can be made up of a single document or can be divided into multiple connected parts. The `$ref` (According to RFC3986) fields in Open API specification is used to reference these parts as follows from the JSON Schema definitions.

The file name `openapi.json` or `openapi.yaml` are the file names recommended for the root OpenAPI document.

5.6.2 YAML

YAML [\[6\]](#) is a human-readable data-serialization format language and it is usually used to store data for configuration files.

YAML intentionally differs from SGML skipping the redundant parts of Extensible Markup Language (XML) and incorporating a minimal syntax to reduce the size of files.

It uses both Python-style indentations to indicate nesting YAML uses python indentations and a more compact format using square brackets [...] for lists and curly brackets {...} for maps.

5.6.3 Swagger

Swagger [\[8\]](#) is an Interface Description Language for describing RESTful APIs expressed using JSON.

Swagger is used for design, build, document and test RESTful web services along with a set of open-source software tools. Swagger includes automated documentation, code generation (into many programming languages), and test-case generation.

Swagger specification was renamed in January 2016 to OpenAPI Specification and was moved to a new software repository on GitHub. While the specification itself was not changed, this renaming signified the split between the API description format and the open-source tooling.

5.6.4 Swagger UI framework

Swagger UI [\[9\]](#) is a web solution that allows visualizing and interacting with the API's resources even when any of the implementation logic it's implemented.

Swagger UI framework is automatically generated from OpenAPI (formerly known as Swagger) specification and provides a visual documentation connecting in a simple way the back-end implementation and the client-side consumption.

5.6.5 OpenAPI Generator (Swagger Codegen)

OpenAPI Generator or Swagger Codegen [\[10\]](#) is an online code generator that supports OpenAPI version 3 specifications [\[11\]](#).



With an OpenAPI Specifications [\[12\]](#), Swagger Codegen allows generation of API client libraries (SDK generation), server stubs and documentation automatically.

5.6.6 Usage in OSLC Bridge

With Swagger Editor the developer can edit a Swagger API specifications in YAML format inside a browser. To preview documentation in real-time, the RESTful APIs for OSLC Bridge is defined as a YAML file of Open API.

Using the full Swagger tooling with a valid Swagger JSON descriptions provides a simple way for code generation, documentation, etc.

Using the docker-compose.yml file to create the container.

```
version: '3.3'
services:
  swagger-editor:
    container_name: swagger-editor
    image: swaggerapi/swagger-editor
    restart: unless-stopped
    ports:
      - 9003:8080
```

Now when connecting to <http://adeptness.ulmaembedded.com:9003> an open API editor is deployed. Part of the OSLC Bridge is developing like a REST interface with this deployed editor.

Open API via Swagger Codegen project allows the generation of API client libraries (SDK generation), server stubs and documentation. The entire set of generated code complies with the OpenAPI Specification which automatically deploys a test environment in port 8080..

The following file is the one used to generate a Java source code for defined REST API for OSLC Bridge:

```
openapi: 3.0.0
info:
  title: OSLC Bridge Microservice REST API
  version: '0.1.6'
  description: REST API for OSLC Bridge Microservice
paths:
  /adms/v1/ping:
    get:
      summary: KeepAlive to know if the service is up
      tags:
        - health
      responses:
        '200':
          $ref: '#/components/responses/StatusOk'
  /adms/v1/info:
```



```
get:
  summary: Get info about the microservice
  tags:
    - health
  responses:
    '200':
      description: Returns the information about the microservice
      content:
        application/json:
          schema:
            $ref: '#/components/schemas/MicroserviceInfo'

/adms/v1/performance:
get:
  summary: Metrics related with the performance of the microservice
  tags:
    - health
  responses:
    '200':
      description: Provides brief information on CPU and memory usage
      content:
        application/json:
          schema:
            $ref: '#/components/schemas/PerformanceInfo'

/adms/v1/status:
get:
  summary: Get microservice status
  tags:
    - status
  responses:
    '200':
      description: Current microservice status
      content:
        application/json:
          schema:
            $ref: '#/components/schemas/Status'
put:
  summary: Change microservice status
  tags:
    - status
  requestBody:
    description: Structure describing the microservice status
    required: true
    content:
      application/json:
        schema:
          $ref: '#/components/schemas/Status'
```



```

    responses:
      '200':
        $ref: '#/components/responses/StatusOk'

/adms/v1/setup:
  post:
    summary: Setup parameters for OSLC Bridge
    requestBody:
      description: Setup parameters for OSLC Bridge
      required: true
      content:
        application/json:
          schema:
            $ref: '#/components/schemas/Setup'
    responses:
      '200':
        description: Setup parameters for OSLC Bridge
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/Setup'
    tags:
      - setup

/adms/v1/notify:
  post:
    summary: Listener for Stellio Subscriptions
    requestBody:
      description: Stellio JSON notification
      required: true
      content:
        text/plain:
          schema:
            type: string
    tags:
      - listener
    responses:
      '200':
        description: Notify?
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/Notify'

components:
  responses:
    StatusOk:
      description: The service is up and respond to ping

```



```
content:
  application/json:
    schema:
      type: object
      properties:
        message:
          type: string
          description: Up or not

BadRequest:
  description: Bad Request
InternalServerError:
  description: Internal error trying to access the resource
  content:
    application/json:
      schema:
        type: object
        properties:
          message:
            type: string
            description: Info about possible error

schemas:
  PerformanceInfo:
    type: object
    properties:
      Memory:
        type: object
        properties:
          Alloc:
            type: number
          TotalAlloc:
            type: number
      CpuLoadAvg:
        type: number
      CpuTime:
        type: number
      CpuAvgUsage:
        type: number

  MicroserviceInfo:
    type: object
    properties:
      id:
        type: number

  Status:
    type: object
    properties:
      status:
```



```

    type: string
    description: Tells if the service is launched but not operating (ready)
or if it is running its functionality (running)
    enum:
      - ready
      - running
      - pause
  Notify:
    type: object
    properties:
      status:
        type: string
        description: Listener for Stellio Subscription
        enum:
          - true
          - false
  Setup:
    type: object
    properties:
      stelliourl:
        type: string
        description: Url of the Stellio server
      stelliouser:
        type: string
        description: Username for Stellio connection
      stelliosecret:
        type: string
        description: Secret for Stellio connection
      stellioauthurl:
        type: string
        description: Url of the Stellio Auth server
      rtcurl:
        type: string
        description: Url of the RTC server
      rtcuser:
        type: string
        description: Username for RTC connection
      rtcpass:
        type: string
        description: Password for RTC connection

```

The previous file is used to define the following seven API REST methods for Adeptness OSLC Bridge.

Method type	Description	Endpoint
GET	Returns the information about the microservice	/adms/v1/info



GET	Provides brief information on CPU and memory usage	/adms/v1/performance
GET	The service is up and responds to ping	/adms/v1/ping
POST	Listener for Stellio Subscriptions	/adms/v1/notify
POST	Setup parameters for OSLC Bridge	/adms/v1/setup
GET	Current microservice status	/adms/v1/status
PUT	Change microservice status	/adms/v1/status

Table 7. Adeptness OSLC Bridge methods.

5.7 Eclipse Lyo

5.7.1 Overview

Eclipse Lyo [\[13\]](#) is an OSLC standard implementation. It is used to promote the adoption of Linked Data principles and the OSLC standards for publishing lifecycle data. Moreover, Eclipse Lyo enables the interoperability of heterogeneous products, services, and other distributed network resources.

The open OASIS OSLC standard is based on a RESTful architecture and Linked Data principles, such as those defined in the RDF family of specifications and the W3C Linked Data Platform.

Eclipse Lyo is an implementation of OSLC standard in Java that supports developers with the development of Java REST-based servers and clients that need to share heterogeneous information as RDF resources.

5.7.2 Lyo SDK

Lyo SDK provides an OSLC Client with a series of full APIs to interact with OSLC Servers. It also provides another layer of functionality built on top of Apache HttpClient and JAX-RS Client. These functionalities allow the developer to manage common use cases such as OAuth handling, form login, sending queries, service discovery, and processing query results.

5.7.3 Usage in OSLC Bridge

The Lyo SDK is the framework used to manage OSLC in Java source. The development of OSLC-Bridge requires the next Lyo dependencies sited in a maven pom.xml file.

Artefact	Version
java	1.8
oslc4j	4.0.0
lyo-client	4.0.0.M2
jena	3.17.0

Table 8. OSLC Bridge dependencies



5.8 Jersey

5.8.1 Overview

Jersey [\[14\]](#) is the reference implementation from Sun (now Oracle) of Jakarta RESTful Web Services, (JAX-RS; formerly Java API for RESTful Web Services) is a Jakarta EE API specification that facilitates the creation of web services according to the Representational State Transfer (REST) architectural pattern.

Eclipse Jersey is a REST framework of JAX-RS (JSR-370) implementation. JAX-RS uses annotations, introduced in Java SE 5, to simplify the development and deployment of web service clients and endpoints.

5.8.2 Usage in OSLC Bridge

The use of the Jersey framework is mainly in `com.ues.adeptness.stellio.utils.StellioUtils` java class using `javax.ws.rs.*` packages.

The Eclipse Lyo uses Jersey internally to manage REST petitions. Jersey is the default framework used in OSLC Bridge to consume REST methods from other resources.

5.9 Spring Boot

5.9.1 Overview

Spring Boot is a way to create stand-alone Spring based Applications, those applications can be deployed and run in several environments.

5.9.2 Spring framework

The Spring Framework is a popular application framework and inversion of control (IoC) container for the Java platform. The core features in the framework's can be used by any Java application, but there are extensions for building web applications on top of the Java EE (Enterprise Edition) platform.

5.9.3 Spring Boot solution

Spring Boot [\[15\]](#) is Spring's coding by convention or convention over configuration solution for creating java based stand alone applications.

5.9.4 Usage in OSLC Bridge

The use of Spring Boot is mainly in `io.swagger autogenerated` package. Spring Boot is the default framework used in OSLC Bridge to define the inner REST methods defined with OpenAPI.

5.10 JUnit

5.10.1 Overview

JUnit is part of the family of unit testing frameworks collectively known as xUnit that originated with SUnit framework. JUnit is focused in provide a unit testing framework for the Java programming language. It has been important in the development of test driven development (TDD).



5.10.2 Usage in OSLC Bridge

Software development teams use unit testing to design robust software elements that maintain code and diminish issues inside code units. It is essential to perceive the importance of identifying and fixing defects in the early stages of the software development life cycle.

The concept of unit testing is used to validate every unit of the software code compared with expectation. This can improve testability and benefits by writing testable code.

For the complete development of OSLC Bridge and its artefacts, the necessary number of test units have been implemented, thus guaranteeing the correct working of the development.

Those unit tests along with other test types are used to test coverage at least for 70% of the source code in OSLC Bridge.

Package	Tests	Quantity
com.ues.adeptness.architecture	Test for maintaining an interfaced architecture.	2
com.ues.adeptness.rest	Tests for REST API.	3
com.ues.adeptness.rtc	Test for RtcUtils.	9
com.ues.adeptness.stellio	Test for StellioUtils.	9
com.ues.adeptness.stellio.responses	Test for StellioUtils in asynchronous mode.	3

Table 9. OSLC Bridge tests

5.11 SonarQube

5.11.1 Overview

SonarQube [\[16\]](#) is an automatic code review solution tool, useful to detect bugs, code smells and vulnerabilities in source code. It can integrate with any existing workflow to enable continuous code inspection across your project branches and changes pull requests.

5.11.2 Usage in OSLC Bridge

SonarQube is used to scan for vulnerabilities, avoid them, and provide quality software.



6 OSLC BRIDGE

6.1 Overview

The OSLC Bridge brings a common standardized interface connected to product lifecycle management tools.

OSLC enables the integration of federated, shared information across tools that support different but related domains.

The most relevant for Adeptness Projects are:

- **Core Specification**, which defines the overall approach to OSLC-based specifications and capabilities. These capabilities are often needed across several domains and provide a solid foundation for reading and writing linked data resources.
- **Configuration** and **Change Management Specification** eases the management of product change requests, activities, tasks and relationships between those, and related resources such as requirements (Requirements Management domain) or test cases (Quality Management domain).
- **Requirements Management Specification** provide the management of requirements, requirement collections and supporting resources defined in OSLC Core specification.
- **Quality Management Specification** defines the test plans, test cases and test results of the software delivery lifecycle. These represent individual resources along with their relationships to other shared resource types such as change requests and requirements.

The main microservice provided by the OSLC bridge is to receive all oracles test case execution results and the conversion of these results in the form of standardized test case execution results and defects.

For this purpose, a REST API-based periodic subscription to the data logger requests the validation results according to the Adeptness validation plan, and the corresponding artefacts are created to be used to adapt the validation plan to the appeared defects at HiL/SiL validation phase.

Features:

- Automatic issue creation with a fail test case verdict
- Compliant with Adeptness interfaces
- Traceability of data obtained during operation validation with lifecycle artefacts

Advantages:

- Reduction on time for defining regression test needed in a new release
- Reduction on time for reproducing bugs appeared in operation
- Improvement in processes or design of next developments due to the collection of test data and its subsequent analysis
- Use of IBM Rational Team Concert along with Jazz Team Server, consolidated tools for managing the OSLC lifecycle

Limitations:



- Correspondence between Adeptness validation model and OSLC model. All data needed for traceability purposes must be included and available in the model (e.g. test case verdict, requirement identifier...)

6.2 Environments

6.2.1 Local Development Environment

The local development environment is based on a Linux distribution (see Table 10), where all the tools deployed in Docker are integrated (see Illustration 10).

Distributor ID	Ubuntu
Description	Ubuntu 18.04.5 LTS
Release	18.04
Codename	Bionic

Table 10. Ubuntu server

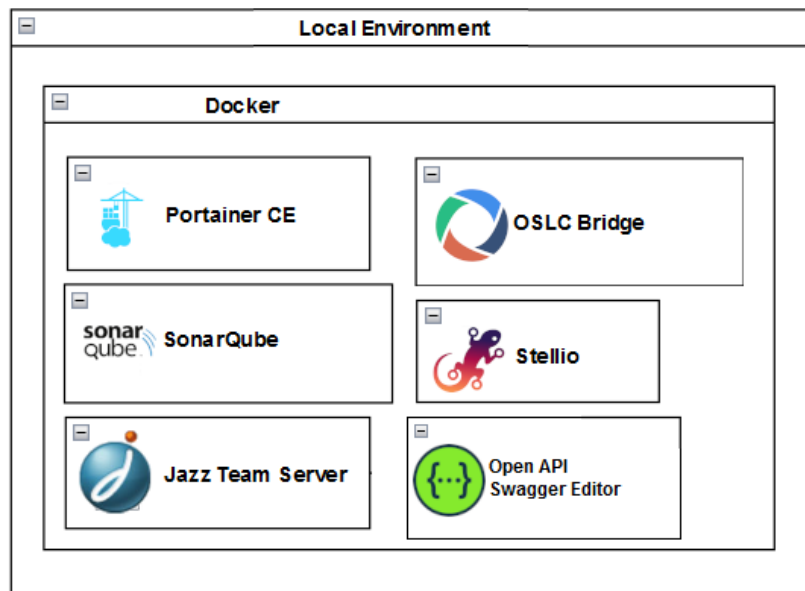


Illustration 10. Local development environment.

6.2.2 Final environment

Docker is deployed in the environment presented in the figure below. It is part of the TUW infrastructure along with the Adeptness toolchain and the IBM Cloud Server.



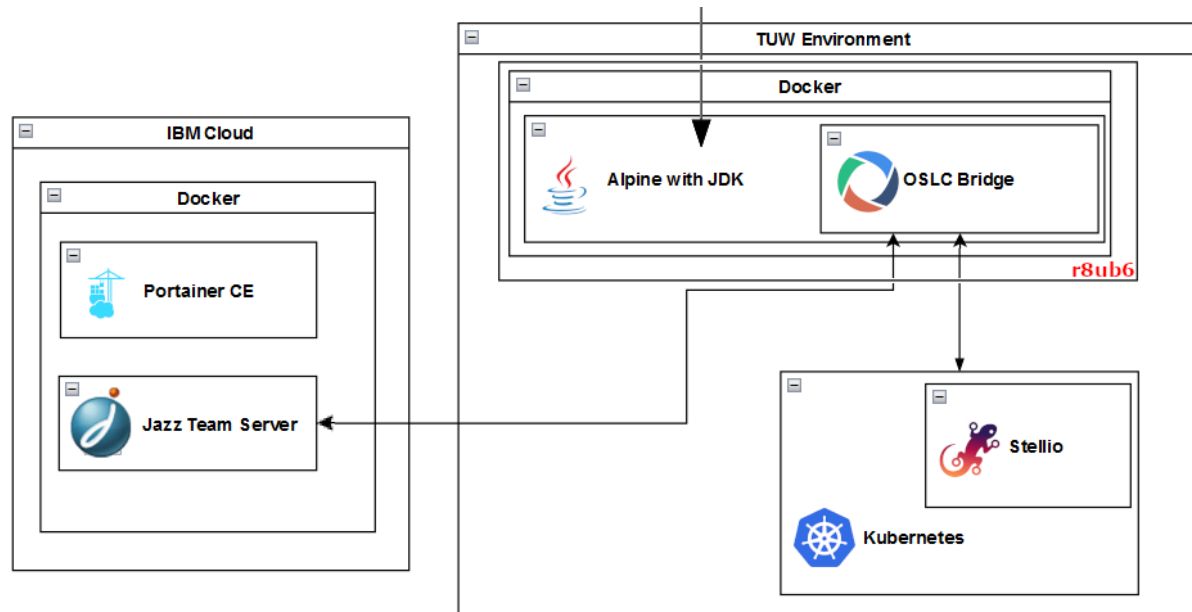


Illustration 11. Final environment.

6.2.2.1 IBM Cloud Demo Environment

The IBM Cloud® platform is a platform as a service (PaaS) and an infrastructure as a service (IaaS) combined to provide an homogenous experience.

The IBM Cloud platform scales and supports with small development teams and organizations and with large enterprise businesses development teams. IBM Cloud provides solutions that enable to deploy locally but with a global scalability.

This environment is used to deploy a Jazz Team Server accessible for all partners. In this environment, a Portainer instance is also deployed to facilitate the administration of the docker containers.

6.2.2.2 TUV Demo Environment

This environment is used to deploy the OSLC-Bridge microservice and connect it to a Stello, that is deployed in this server too. The Stello server with the Adeptness Validation model is deployed in TUV via Kubernetes and it is accessible in IP 128.130.123.122 port 31312

6.3 Artefacts

To develop the final program, several artefacts have been developed, the purpose of these is decoupling dependencies between packages, maintain a well structured system, and allow future development.

Artefact	Definition	Type
RTCUtils	Utilities for managing Rational Team Concert with a simple WorkItem class.	Library
RTCFactory	Different factories to access to OSLC data as Legacy, Jena and DOM4j	Library



StellioUtils	Library to manage Stellio model with Java classes	Library
StellioTestServer	Library for mockup a Stellio server for testing.	Library
RtcTestServer	Library for mockup a RTC server for testing.	Library

Table 11. Developed artefacts

6.4 Architecture

The development is divided into two decoupled codes as shown in the figure below. One is the definition using OpenApi, spring.io package; and the other is the Business Logic of the Adeptness Bridge.

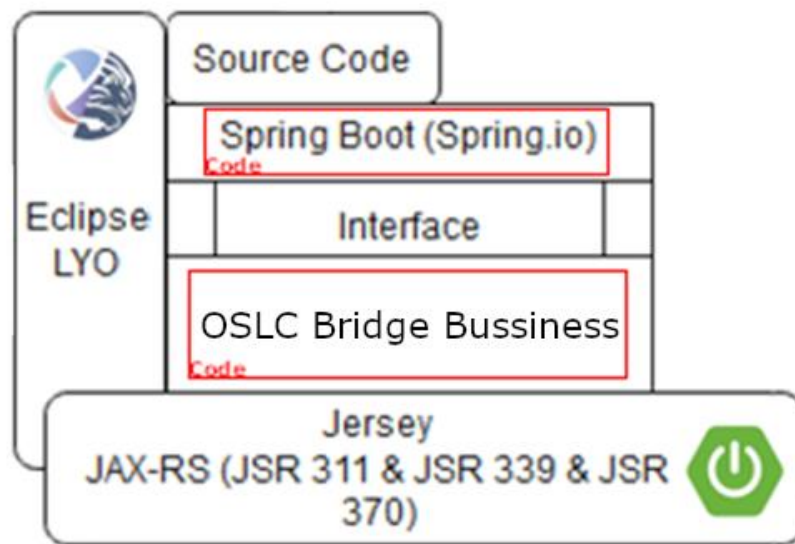


Illustration 12. OSLC Bridge architecture.

The following sections provide more detail of the main architecture block presented in the figure above.

6.4.1 Spring Boot (Spring.io)

Open API via Swagger Codegen project allows generation of API client libraries (SDK generation), server stubs and the documentation of OpenAPI Specifications. With the previous file, a Java source code for the defined REST API is generated.

6.4.2 Interface

An interface has been developed to decouple business logic from the Rest API implementation. This allows OSLC Bridge Business logic to be independent of the Rest Open API definition.

```
/**
 * Interface for Adeptness controller.
 */
public interface AdeptnessControllerInterface {

    public ResponseEntity<MicroserviceInfo> admsV1InfoGet();
}
```



```

public ResponseEntity<Setup> admsV1SetupPost(Setup body);

public ResponseEntity<PerformanceInfo> admsV1PerformanceGet();

public ResponseEntity<InlineResponse200> admsV1PingGet();

public ResponseEntity<Status> admsV1StatusGet();

public ResponseEntity<InlineResponse200> admsV1StatusPut(Status body);

public ResponseEntity<Notify> admsV1NotifyPost(String body,
        HttpServletRequest request);
}

```

6.4.3 OSLC Bridge Business

Adeptness Business involves the implementation of several artefacts (e.g., java utils libraries) on the Adeptness domain (NGSI-LD validation model) for an OSLC standard integration.

6.4.3.1 RTCUtils

Eclipse Lyo client in a ChangeRequest class encapsulates work item of OSLC standard. For our purposes, the use of ChangeRequest is to abstract for mapping concepts of NGSI-LD to OSLC RDF.

RTCUtil is a library developed to manage the OSLC Client via Lyo Client with the Workitem class called RtcWorkItem. This class has the minimum to meet the needs of requirements avoiding the complexities of Lyo Client in a single type of class and helped methods.

Some of the methods that this class contains are:

Method name	Description
login()	Login in JTS Server
createWorkItem(RtcWorkItem)	Create an RtcWorkItem
createDefect(RtcWorkItem)	Create an RtcWorkItem of type Defect
createTask(RtcWorkItem)	Create an RtcWorkItem of type Task
getProjectAreas()	Get all Project áreas in this RTC Server
getWorkItemById(URL, String)	Get an RtcWorkItem by ID and Project Area
getRdfData(URL)	Get the RDF data from URL
getUrlFromId(String, String)	Get the URL link for a ID
query(String, String, String)	Query with select and where againts JTS Server



queryOslcQuery(String, String, String)	Query with OSLC Query againsts JTS Server
updateWorkItem(RtcWorkItem)	Update an RtcWorkItem of any type.
logout()	Logout in JTS Server

Table 12. RTCUtils methods

6.4.3.2 RTCFactory

In addition to Lyo Client, there are other alternatives to access, generate or process RDF data from and to OSLC Server.

There are different Java factories, as it is shown in the [Artifacts section](#), to access OSLC data as Legacy, Jena and DOM4j. The developer only has to select one of them and will have an abstraction for RTCUtils using the desired implementation.

Factory name	Library used	Link
Legacy	Implemented using Lyo Client	https://oslc.github.io/developing-oslc-applications/eclipse_lyo/eclipse-lyo.html
Apache Jena	Implemented using JENA Client	https://jena.apache.org/index.html
DOM4J	Create an RtcWorkItem of type Defect	https://dom4j.github.io/

Table 13. OSLC Bridge Factories

6.4.3.3 StellioUtils

Analogously to the previous library made for Rational Team Concert (RTCUtils), a JAVA library has been done for NGSI-LD Context Broker (Stellio) which is called StellioUtils.

At this time, this library is only for internal use. In next releases, the possibility of make it open-source will be analyzed.

To conform to the requirements, and to facilitate development, the interaction with Stellio is abstracted to a few methods contained in this library.

Method name	Description
login()	Login in Stellio Server.
createSubscription(String)	Create a Subscription.
getStellioEntity(Class<?>)	Get a Stellio entity by defined Java class.
getStellioEntityById(String)	Get a Stellio entity by unique ID.
getStellioEntityByIdAsync(String, StellioResponseReceivedAsyncInterface)	Get a Stellio entity by unique ID asynchronously.



getStellioEntityByType(Class<?>)	Get a Stellio entity by type defined in a Java class.
getStellioEntityByType(String)	Get a Stellio entity by type as String.
getStellioEntityByTypeAsync(Class<?>, StellioResponseReceivedAsyncInterface)	Get a Stellio entity by type defined in a Java class asynchronously.
getStellioEntityByTypeAsync(String, StellioResponseReceivedAsyncInterface)	Get a Stellio entity by type as String asynchronously.
logout()	Logout from Stellio Server.

Table 14. StellioUtils methods.

Along with StellioUtils, a StellioFollowEngine is developed to traverse the graph with the method followRelations.

Method name	Description
followRelations(StellioEntity)	Follow the relations with Stellio Object from a Stellio Entity.
followRelations(StellioEntity, Class<?>)	Follow the relations with StellioObject from a Stellio Entity, only return classes instance of a class.

Table 15. followRelations method

To follow relations from all StellioObject to a list of defined StellioEntity, all methods are inspected with reflection recursively and all StellioEntities are traversed to find the listed StellioObjects.

6.4.3.4 Test Servers

Test units suffer testing problems when making tests against the URLs, for this reason and aiming at solving it, two test libraries have been developed:

Both libraries mock the servers with a java mocking library, allowing test fake responses from the server simulating a true server.

6.4.3.4.1 RtcTestServer

To allow testing the RTCUtils library, we created another library for testing, called RTCTestServer. This library uses a MockServer with expectations defined for RTC.

The expectations are defined for each OSLC element needed such as ProjectAreas, Catalog, Defects, Literals, Queries, WorkItems, etc.

6.4.3.4.2 StellioTestServer

To allow testing the StellioUtils library, we created another library for testing, called StellioTestServer. This library uses a MockServer with expectations defined for Stellio.

The expectations are defined for Stellio elements needed such as Entities and queries by type.



6.5 Validation Model

6.5.1 Overview

The Adeptness validation model is used in OSLC Bridge implementation to subscribe to a test case entity and to receive a notification when a test case fails. This test case entity is mapped to the OSLC Defect work item to create an issue when an error occurred. In the current model, the test case entity includes the verdict property.

In addition, a new requirement entity has been created. This entity has been linked to the test cases entity, thus providing complete traceability of the possible issues.

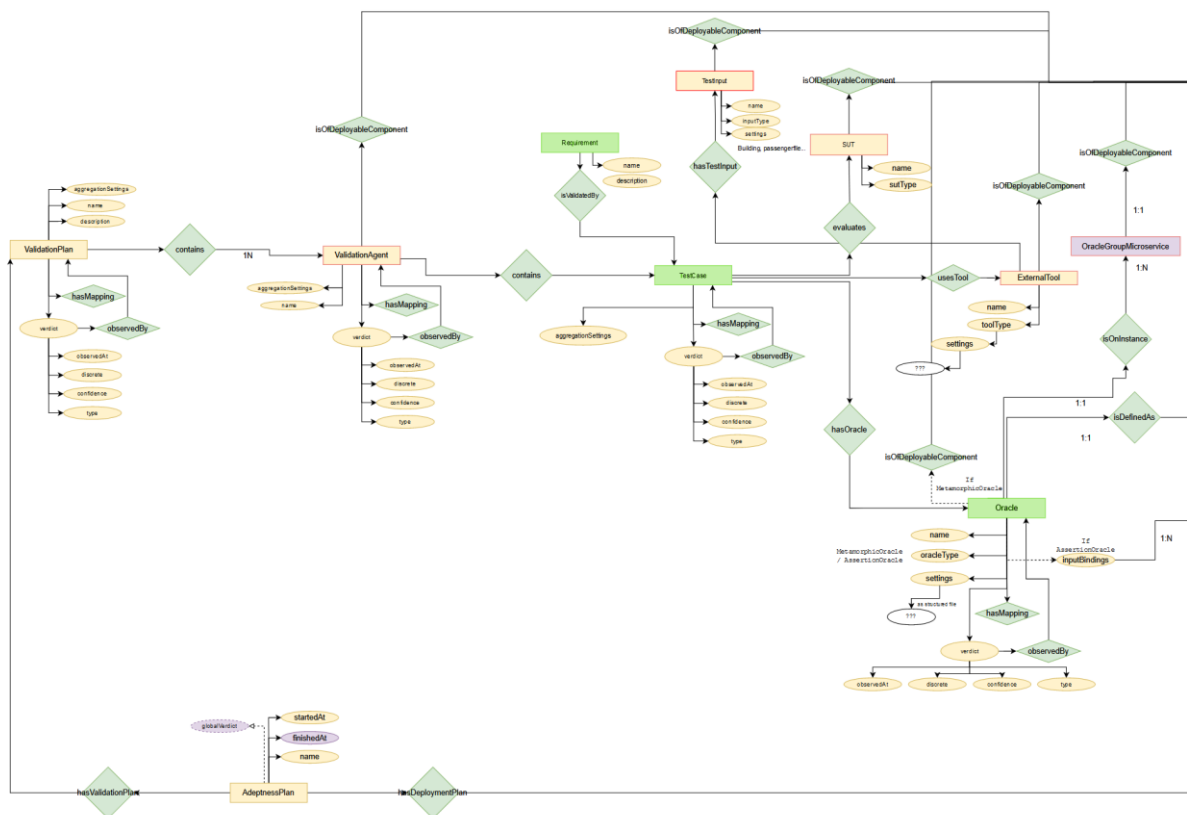


Illustration 13. Validation model.

6.5.2 Entities

Among all this model, the entities created for traceability are zoomed in the next figure.

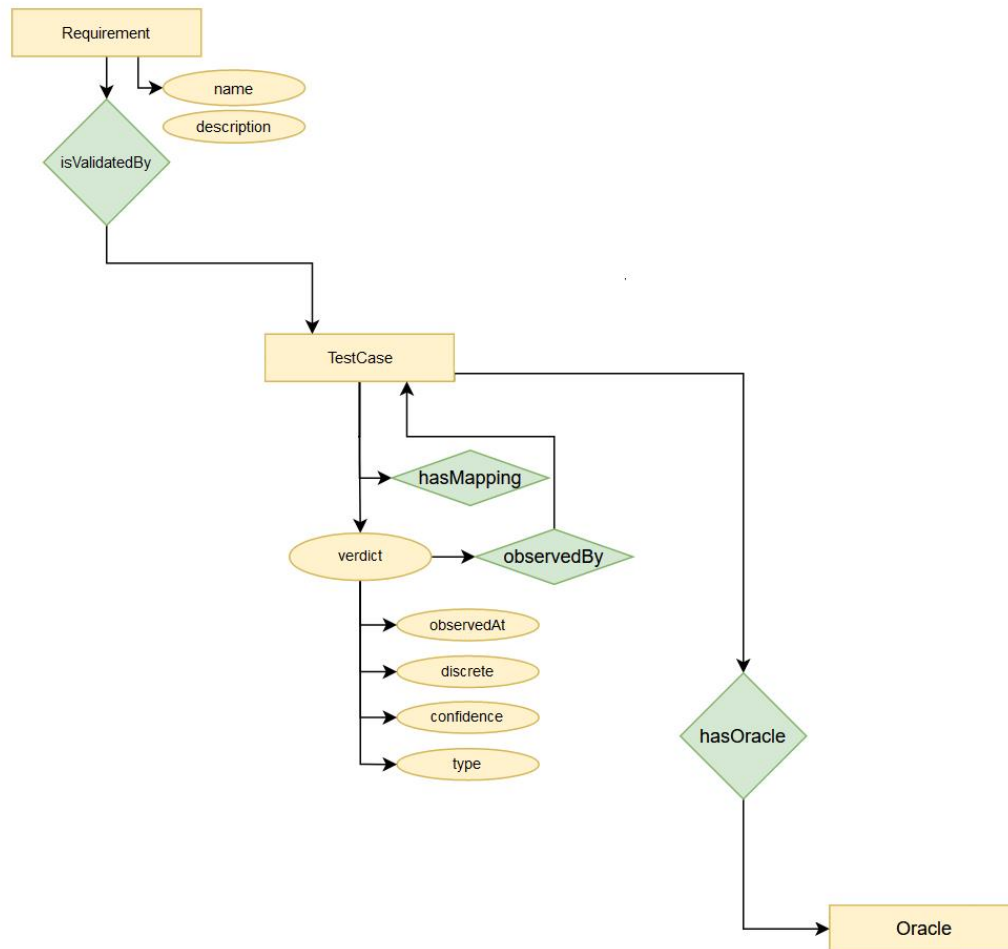


Illustration 14. Testcase with verdict as parameter.

The following table maps the validation model entities and the OSCL resources in the OSLC Bridge.

Entity	URN namespace	OSCL Domain
Requeriment	urn:ngsi-Id:Requeriment:IDSTRING	Requirement resource in RM domain
Test Case	urn:ngsi-Id:TestCase:IDSTRING	Defect resource in CCM domain

Table 16. Mapping Adeptness model with OSLC Domain

To allow a complete traceability, in next releases the OSLC Bridge will add a new OSLC Domain, as it is shown in the following table.

Entity	URN namespace	OSCL Domain
Requeriment	urn:ngsi-Id:Requeriment:IDSTRING	Requirement resource in RM domain
Oracle	urn:ngsi-Id:Oracle:IDSTRING	Defect resource in CCM domain
Test Case	urn:ngsi-Id:TestCase:IDSTRING	Test Case resource in QM domain

Table 17. Mapping Adeptness model with OSLC Domain



More information about the traceability in [Section 6.6.3](#)

6.6 Operation

6.6.1 Setup

As long as the OSLC Bridge is not configured, it will remain inactive until it is activated through the setup method. When setup method is called, the OSLC Bridge begin the process of executing the next action:

- Login in RTC Server
- Login in Stellio Server
- Create a subscription to Stellio Server
- Awaiting notification from Stellio Server

To login RTC correctly, next parameters are needed:

Setup Parameter	Description
rtc.url	RTC Server address.
rtc.user	RTC user for login.
rtc.password	RTC password for login.
rtc.projectarea	Project Area where information of the project is stored.
rtc.context	RTC context in JTS.

Table 18. Setup RTC.

These parameters can be sent in the setup method as arguments or defined in properties files on the server-side.

The rtc.projectarea parameter needs that a Project Area in RTC for Adeptness project has been previously created. A project area defines the project structure, project deliverables, team structure, schedule and process for a project

To login Stellio correctly, next parameters are needed:

Setup Parameter	Description
stellio.url	Stellio Server address.
stellio.auth.url	Stellio Auth Server address.
stellio.entities.path	Path to entities ngsi-ld.
stellio.client.id	Stellio user for login.
stellio.client.secret	Stellio password for login.

Table 19. Setup Stellio.



Like the previous parameters, these can be sent in the setup method as arguments or defined in properties files on the server-side.

When the two login processes are correct, a subscription of Stellio is generated, and a Stellio subscription against TestCase entities in the validation model is created.

The Stellio subscription needs a query to create a proper subscription. The query works subscribing to changes in Verdict property of TestCase entities, and it triggers when the discrete property of verdict property is Failed.

Receiving from the server a response that the correct subscription has been created, the OSLC Bridge awaiting until a notification will be received.

6.6.2 Notification

At some point, an external agent with no relation with OSLC Bridge will change the value of discrete verdict to fail. Then a notification from Stellio is fired. The endpoint for that notification is a REST method in OSLC Bridge development.

When the fired notification is received in the OSLC Bridge, the OSLC Bridge begin with the creation of an OSLC Artifact of type Defect into the CCM domain.

To create the defect, the work item OSLC Bride collects information from Stellio Server, using, if necessary, the developed StellioFollowEngine to traverse the graph.

With the collected information, the OSLC Bridge maps concepts from NGSI-LD Adeptness validation model to OSLC Standard

The Defect created contains the next attributes, both mandatories and optionals, in an RDF file.

Attributtes	Description
dcterms:created	Dublin Core, Timestamp of resource creation.
dcterms:type	Literal for Defect
rtc_cm:filedAgainst	Workitem category for which a work item is filed against for.
rtc_cm:timeSheet	Starting date of a work item time entry.
dcterms:contributor	Dublin Core, responsible for Change Request.
rtc_cm:resolvedBy	User that resolved the work item.
rtc_cm:state	Status of the work item
oslc_cm:verified	Change Request has been verified.
oslc:discussedBy	Comments and notes of the Change Request.



acc:accessContext	Access Context for IndexableLinkedDataProvider.
oslc_cm:project	In the Extended Change Request Definition namespace, the project of this Change Request.
dcterms:creator	Dublin Core, Creator of the resource.
process:projectArea	Link a resource to its project area.
rtc_cm:modifiedBy	User that make the last modification to the work item.
dcterms:modified	Dublin Core, Timestamp of resource modification.
dcterms:identifier	Unique identifier for a resource.
oslc_cm:testedByTestCase	Test case by which this change request is tested.
oslc_cm:approved	Change Request has been approved.
rtc_cm:progressTracking	Progress information for a work item.
oslc_cm:inprogress	Change Request is in an active work state.
oslc_cm:fixed	Change Request has been fixed.
oslc:instanceShape	Resource Shape for resource property value-types and allowed values.
oslc:shortId	Like dcterms:identifier but in shorter form.
rtc_cm:repository	Repository of the work item.
rtc_cm:type	Type of the work item. A Defect type Workitem.
dcterms:description	Dublin Core is a sufficiently descriptive text.
oslc_cm:closed	Change Request is done
oslc:shortTitle	Short title name to identify a resource
oslc:serviceProvider	OSLC Service Provider link.
rtc_cm:subscribers	Subscribers of the work item.
dcterms:subject	Tag for a resource
oslc_cm:priority	In the Extended Change Request Definition namespace, the priority of this Change Request.
oslc_cm:severity	In the Extended Change Request Definition namespace, the severity of this change request.



rdf:type	rdf:resource="http://open-services.net/ns/cm#ChangeRequest"
oslc_cm:status	The status of the Change Request.
oslc_cm:reviewed	Denote if Change Request has been reviewed.
dcterms:title	Dublin Core, short summary, single line.

Table 20. OSLC attributes.

Steps performed by the OSLC Bridge to perform the creation of a defect work item are:

- Receiving notification from Stellio Server
- Collect information from Stellio Server about TestCases.
- Map NGSI-LD to an OSLC RDF Resource
- Create a defect work item.

6.6.3 Traceability

The Verdict property contains references to the associated TestCase, and thus, to the validated requirement through the associated link to Requirement Entity. This can be seen in the semantic model stored defined for Stellio:

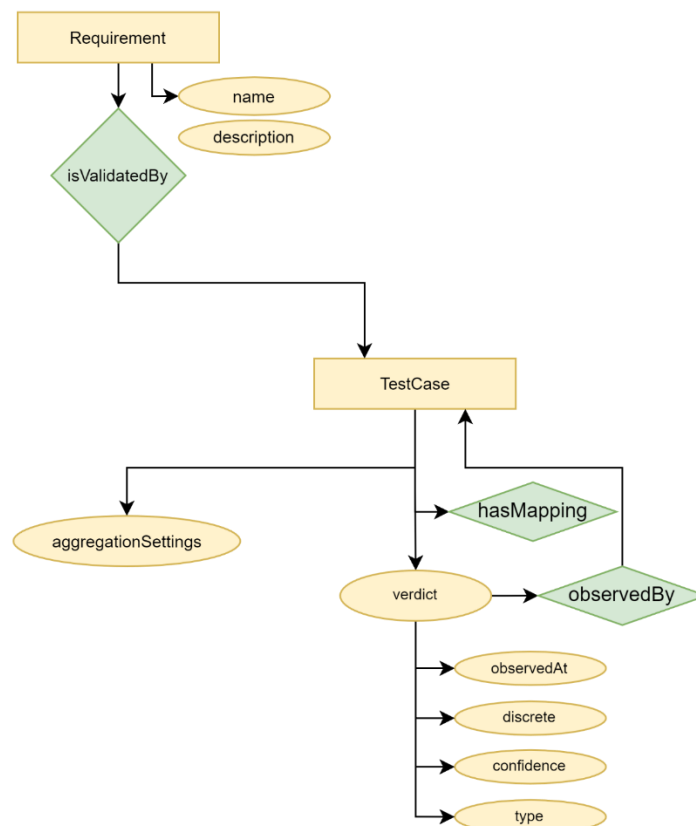


Illustration 15. Testcase based model.



That means, specifically, creating a defect from the context information received from Stellio in NGSI-LD format.

- The reference to TestCase Entity in NGSI-LD can be mapped to Requirements Management through OSLC Bridge, using OSLC QM TestCase artefact, through the Change Request artefact to which is associated the Defect.
- The reference to Requirement can be tracked back to Requirements Management domain in OSLC or to the `oslc_cm:affectsRequirement`, `oslc_cm:implementsRequirement`, or `oslc_cm:tracksRequirement` properties in Defect artifact.

The OSLC Bridge project is a Spring Boot Web Service generated via OpenAPI in a Maven project. This Maven project admits goals like `spring-boot:run` or `clean install spring-boot:run`

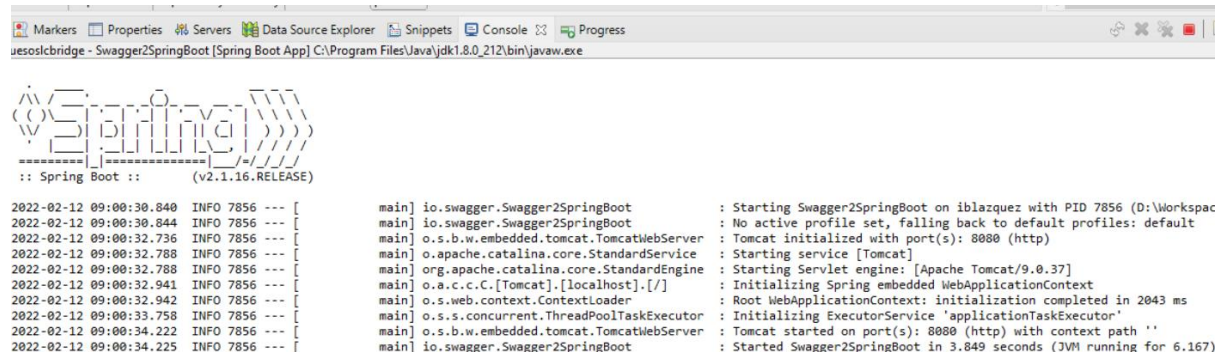
```
$ mvn spring-boot:run
```

Illustration 16. OSLC Bridge running from CLI.



```
$ mvn clean install spring-boot:run
```

From eclipse, with Spring Tools for Eclipse IDE installed, run from Eclipse with Run As > Spring Boot App.



```

2022-02-12 09:00:30.840 INFO 7856 --- [main] io.swagger.Swagger2SpringBoot : Starting Swagger2SpringBoot on iblazquez with PID 7856 (D:\Workspac
2022-02-12 09:00:30.844 INFO 7856 --- [main] io.swagger.Swagger2SpringBoot : No active profile set, falling back to default profiles: default
2022-02-12 09:00:32.736 INFO 7856 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 8080 (http)
2022-02-12 09:00:32.788 INFO 7856 --- [main] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2022-02-12 09:00:32.788 INFO 7856 --- [main] org.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/9.0.37]
2022-02-12 09:00:32.941 INFO 7856 --- [main] o.s.c.c.c.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext
2022-02-12 09:00:32.942 INFO 7856 --- [main] o.s.web.context.ContextLoader : Root WebApplicationContext: initialization completed in 2043 ms
2022-02-12 09:00:33.758 INFO 7856 --- [main] o.s.s.concurrent.ThreadPoolTaskExecutor : Initializing ExecutorService 'applicationTaskExecutor'
2022-02-12 09:00:34.222 INFO 7856 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8080 (http) with context path ''
2022-02-12 09:00:34.225 INFO 7856 --- [main] io.swagger.Swagger2SpringBoot : Started Swagger2SpringBoot in 3.849 seconds (JVM running for 6.167)

```

Illustration 17. OSLC Bridge running from IDE.

To create JAR package with maven:

```
$ mvn package
```

To dockerize:

- Create the JAR
- Copy the JAR file adeptness-oslbridge.jar in target/ to docker/jar
- Into the docker path for create image run:

```
$ docker build . -t ues/adeptness-oslbridge:0.1.0
```

- To run the image, creating a container named adeptness-oslbridge:

```
$ docker run -d --name adeptness-oslbridge -p:8080:8080 ues/adeptness-oslbridge:0.1.0
```

6.8 Deploy

To deploy:

- Install docker from docker.io <https://docs.docker.com/engine/install/ubuntu/>
- Dockerize the OSLC Bridge copying the JAR file adeptness-oslbridge-0.1.5.jar in target/ to docker/jar
- Using this Dockerfile

```

FROM adoptopenjdk/openjdk11:jre-11.0.6_10-alpine
LABEL version="0.1" maintainer="Sergio Rodriguez
<srodriguez@ulmaembedded.com>"
COPY jar /home
#If configs files are created put directly into docker
#COPY stellio.properties /

```



```
#COPY rtc.properties /
#COPY adeptness.properties /
CMD java -jar /home/adeptness-oslcbridge-0.1.5.jar
io.swagger.Swagger2SpringBoot
EXPOSE 8080
```

- Create image for Docker run:

```
sudo docker build . -t ues/adeptness-oslcbridge:0.1.5
```

- To run the image, creating a container named adeptness-oslcbridge

```
sudo docker run -d --name adeptness-oslcbridge-rtc -p:8080:8080 --add-host
adeptness.ulmaembedded.com:172.33.0.2 ues/adeptness-oslcbridge:0.1.5
```

- OSLC Bridge needs connections with Stellio network, an internal network and Jazz Team Server network; and external network in the IP 172.33.0.2 for adeptness.ulmaembedded.com that contains an IBM Cloud server where is deployed IBM Jazz Team Server for adeptness.
- Connect to the docker container and change or create the properties files to work in this environment.

OSLC Bridge configuration

```
## PAY ATTENTION this file overwrite config from
/uesoslcbridge/src/main/resources JAR
oslc.bridge.about=(c) 2021 Ulma Embedded Solutions
oslc.bridge.version=v 0.1.5
oslc.bridge.ip=http://128.130.123.118:8080/adms/v1/notify
oslc.bridge.time=HH:mm:ss
```

For RTC

```
rtc.context=ccm/
rtc.url=https://public.adeptness.ulmaembedded.com:9443/
rtc.user=adeptness
rtc.password=*****
rtc.projectarea=adeptness (Change Management)
```

For Stellio

```
stellio.entities.path=/ngsi-ld/v1/entities
#stellio.auth.url=https://sso.eglobalmark.com/auth/realms/stellio
-dev/protocol/openid-connect/token
```




```
stellio.url=http://128.130.123.122:31312  
stellio.client.id=adeptness  
stellio.client.secret=*****_***_***_***
```

- Now the bridge is ready to start using REST API method.

```
curl -X POST "http://128.130.123.118:8080/adms/v1/setup" -H "accept:  
application/json" -H "Content-Type: application/json" -d "{}"
```



7 SUMMARY

In the CPSoS context, it is important to know which lifecycle artefacts are affected by situations that occurred at operation time to reduce the time spent in impact and traceability analysis. For this purpose, we focused on the trace of the operational data and artefacts, which has been accomplished using the OSLC Bridge generated in Adeptness.

The OSLC Bridge is an Adeptness-specific implementation bringing together OSLC world and Adeptness NGSI-LD defined entities, microservices, attributes and resources. The overall ecosystem does not only manages the CPSoS seamless interoperability, but also enables lifecycle collaboration, from requirements management, and design, to verification and validation artefacts.

This collaboration provides a complete picture of product lifecycle management and application lifecycle management and eases traceability from conception to runtime execution.



8 RISK REGISTER

Risk Number	Description of Risk	Proposed Risk Mitigation Measure	Probability/ effect
1	Incompatibility: The solution adapters do not use OSLC- supported applications for RM, QM or CCM	The source code decoupled helps to mitigate this risk. However, some source code needs to be adapted.	High
2	Dependency of commercial OSLC solutions	As partners of IBM, we could put in contact with the industrial partners with IBM solutions	Medium
3	High resource consumption	The technical requirements can be nowadays easily overcome with an upgrade of corresponding server memory, hard disk, etc.	Low



9 REFERENCES

- [1] [https://www.oasis-open.org/committees/download.php/61054/Oslc Core 2.0 Final.pdf](https://www.oasis-open.org/committees/download.php/61054/Oslc%20Core%202.0%20Final.pdf)
- [2] <https://www.eclipse.org/lyo/>
- [3] <https://www.w3.org/RDF/>
- [4] <https://docs.docker.com/get-started/>
- [5] <https://www.ibm.com/docs/es/elm/6.0?topic=overview-rational-team-concert>
- [6] <https://www.ibm.com/docs/ro/elm/6.0.6.1?topic=capabilities-rational-team-concert>
- [7] <https://www.egm.io/>
- [8] <https://yaml.org/>
- [9] <https://swagger.io/specification/>
- [10] <https://github.com/swagger-api/swagger-ui>
- [11] <https://github.com/swagger-api/swagger-codegen>
- [12] <https://graphql-faas.github.io/OpenAPI-Specification/IMPLEMENTATIONS.html>
- [13] https://oslc.github.io/developing-oslc-applications/eclipse_lyo/eclipse-lyo.html
- [14] <https://eclipse-ee4j.github.io/jersey/>
- [15] <https://spring.io/projects/spring-boot>
- [16] <https://www.sonarqube.org/>



10 QUALITY ASSURANCE

The executive board is the body for quality assurance. The procedure for review and approval of deliverable is described in Deliverable Report D8.1 – “Project handbook”. The quality will be ensured by checks and approvals by WP Leaders as part of the executive board (see front pages of all deliverables).



11 ACKNOWLEDGEMENTS



This project has received funding from the European Union's Horizon 2020 research and innovation programme under Grant Agreement no. 871319.

Disclaimer

This document reflects the views of the author(s) and does not necessarily reflect the views or policy of the European Commission. Whilst efforts have been made to ensure the accuracy and completeness of this document, the ADEPTNESS consortium shall not be liable for any errors or omissions, however, caused.

