BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF INTELLIGENT SYSTEMS
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

UNIFIED ACCESS TO REQUIREMENTS IN JIRA

JEDNOTNÝ PŘÍSTUP K POŽADAVKŮM V NÁSTROJI JIRA

BACHELOR'S THESIS
BAKALÁŘSKÁ PRÁCE

AUTHOR **DANIEL PINDUR**
AUTOR PRÁCE

SUPERVISOR **Ing. JAN FIEDOR, Ph.D.**
VEDOUCÍ PRÁCE

BRNO 2023

BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY

Bachelor's Thesis Assignment

148230

Institut:	Department of Intelligent Systems (UITS)
Student:	**Pindur Daniel**
Programme:	Information Technology
Specialization:	Information Technology
Title:	**Unified Access to Requirements in JIRA**
Category:	Web applications
Academic year:	2022/23

Assignment:

1. Study the tools for developing OSLC adapters (e.g., Eclipse Lyo) and the Requirements Management for Jira (R4J) app.
2. Study the OSLC Requirements Management (RM) specification.
3. Design an OSLC adapter providing RM-compliant information about requirements.
4. Implement the OSLC adapter to provide RM-compliant access to the requirements in the R4J app.
5. Demonstrate the functionality of the implemented adapter by accessing and editing requirements in one or more R4J projects through the adapter's OSLC interface.

Literature:

- OSLC: https://open-services.net/
- OSLC Requirements Management specification: https://docs.oasis-open-projects.org/oslc-op/rm/v2.1/os/requirements-management-spec.html
- Eclipse Lyo: https://www.eclipse.org/lyo/
- R4J: https://marketplace.atlassian.com/apps/1213064/r4j-requirements-management-for-jira

Requirements for the semestral defence:
First three items of the assignment.

Detailed formal requirements can be found at https://www.fit.vut.cz/study/theses/

Supervisor:	**Fiedor Jan, Ing., Ph.D.**
Head of Department:	Hanáček Petr, doc. Dr. Ing.
Beginning of work:	1.11.2022
Submission deadline:	10.5.2023
Approval date:	3.11.2022

Abstract

The goal of this work is to create an interface for accessing requirements resources stored in Jira Software with the Requirements for Jira (R4J) plugin using an OSLC adaptor for the Requirements Management specification. The adaptor has been split into two to explore the possibility of not needing the R4J plugin – one for the Jira Software (satisfying all requirements of the Requirement Management specification) and one for the R4J plugin (providing additional functionality). Both adaptors have been implemented by utilizing the Eclipse Lyo tooling and the OSLC4J library. The basic overview of the fundamental technologies of OSLC, OSLC Core and Requirement Management specifications, Jira and R4J is provided. The thesis contains a detailed summary of the adaptors' design, implementation, and testing process, as well as an evaluation of the results and used technologies.

Abstrakt

Cílem této práce je vytvořit rozhraní pro přístup k požadavků uloženým v Jira Software s modulem Requirements for Jira (R4J) s využitím OSLC adaptéru pro specifikaci Správa požadavků. Adaptér byl rozdělen na dvě části, z důvodu vyhodnocení zda je možné splňovat požadavky i bez modulu R4J – jeden pro Jira Software (splňující všechny požadavky specifikace Správa požadavků) a druhou pro modul R4J (poskytující další funkce). Oba adaptéry byly implementovány s využitím nástrojů Eclipse Lyo a knihovny OSLC4J. Je uveden základní přehled základních technologií OSLC, specifikací OSLC Core a Správa požadavků, Jira a R4J. Práce obsahuje podrobné shrnutí procesu návrhu, implementace a testování adaptérů a vyhodnocení výsledků a použitých technologií.

Keywords

OSLC, OSLC Adaptor, OSLC Requirement Management, Jira, Requirements for Jira, R4J, Eclipse Lyo, RDF, Linked Data, REST

Klíčová slova

OSLC, OSLC Adaptér, OSLC Správa Požadavků, Jira, Requirements for Jira, R4J, Eclipse Lyo, RDF, Propojená Data, REST

Reference

PINDUR, Daniel. *Unified Access to Requirements in JIRA*. Brno, 2023. Bachelor's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Ing. Jan Fiedor, Ph.D.

Unified Access to Requirements in JIRA

Declaration

I hereby declare that this Bachelor's thesis was prepared as an original work by the author under the supervision of Ing. Jan Fiedor, Ph.D. I have listed all the literary sources, publications and other sources, which were used during the preparation of this thesis.

.....
Daniel Pindur
May 9, 2023

Acknowledgements

I would like to thank my supervisor Ing. Jan Fiedor, Ph.D., for his guidance and advice, which were invaluable during the preparation of this thesis. Special thanks go to my family and friends for their support while working on this thesis and during my studies at BUT.

Contents

Chapter 1

Introduction

Software development is a complex process that requires careful planning and management. Companies often use various software applications to support the development process, such as issue tracking, version control, or requirement management systems. However, this creates a new problem – the integration of these applications. This thesis focuses on the integration of two such applications used for *Requirement Management*, *Jira Software* and *Requirements for Jira* (R4J) plugin, using the Open Services for Lifecycle Collaboration (OSLC) standard.

1.1 Motivation and Objectives

Requirements Engineering is a crucial part of the software development process. As such, the management of the requirements is supported by various software applications called *Requirement Management Systems*, out of which the most notable are IBM Doors and R4J. It is often necessary to integrate these systems with other applications used in the development process. *IBM Doors NG* offers a built-in OSLC interface for this purpose, but no such support is provided in R4J. This significantly limits the possibility of migration from IBM Doors to the R4J solution, as the companies often use the OSLC interface for integration – e.g., Honeywell International Inc. uses the OSLC interface to provide its clients with access to requirements. The goal of this thesis is to explore and create a solution that will allow access to requirements stored in Jira R4J via the OSLC interface for Requirement Management Specification.

1.2 Solution

There are two ways to add OSLC support to a web application, either as an add-on or a standalone web application. After careful consideration, a decision was made to create the adaptor as a standalone web application. The main reason for this decision was to reduce the coupling between the web application and the OSLC interface, which allows for easier maintenance and development, as well as the possibility of using the OSLC adaptor with other web applications, besides Jira R4J, in the future.

During the initial design phase, it was also decided to split the adaptor into two separate adaptors. One for Jira, responsible for the main functionality specified by the Requirement Management Specification, and one for R4J, providing additional functionality, such as the ability to create folders and link requirements to folders. This decision was made in order

to explore the possibility of not needing to use the R4J plugin at all and instead using Jira directly. It also leads to the final solution being generic and less coupled to a specific web application, allowing for easier enhancement or replacement of the underlying web application in the future.

Both adaptors were created using Eclipse Lyo, a project containing SDKs and other utilities used for easier development of OSLC applications. Lyo Designer was used to create the Domain and Toolchain models, which represent the data and capabilities of both adaptors. These models were then fed to Code Generator to generate the code skeletons, compliant with the OSLC specification, for the adaptors. The generated skeletons were then filled with the actual code, implementing the functionality of the adaptors and extended with additional functionality, such as OAuth2 authentication.

1.3 Results

Both adaptors were created as standalone REST-based Java web applications built on the Maven framework. During the development, a collection of HTTP requests was created in Postman, detailing the example usage of created adaptors. The functionality of both adaptors was verified by end-to-end testing, utilizing the Postman Test utility. For further verification of the base capability of Requirement Management Specification, implemented in Jira adaptor, a basic Python client was developed, providing the option to download and upload all of the requirements, in the specified project, in the ReqIF format.

The mapping of the data provided by Jira and R4J API to the OSLC format was done as generically as possible to accommodate all possible use cases, which differ significantly between different companies. During the development, some issues and missing functionality were discovered in the R4J API, resulting in the non-optimal implementation of some of the functionality of the R4J adaptor.

The utilities and SDKs provided by Eclipse Lyo were very useful in the development and design stages of the adaptors, greatly reducing the time needed to adopt the OSLC standard. However, the documentation and examples provided by the Lyo project were outdated and not detailed enough, also lacking some deeper explanation of important concepts. The contents of the Requirement Management Specification were also not documented sufficiently, compared to, for example, the Automation Domain Specification, resulting in difficulties in the comprehension of the specification.

1.4 Structure of the Thesis

This section provides a brief overview of the structure of the thesis itself. Chapter 2 covers the basics of requirements management and its fundamental concepts, as well as a basic overview of requirement management systems. Chapter 3 provides a detailed description of the OSLC standard, its concepts, the Core and Requirement Management specification, and the tooling provided by the Lyo project. In production environments, security is a crucial part of any application. The OSLC interface has to be secured not to allow unauthorized access to the data. This is done by using the BASIC and OAuth2 authentication methods, which are described in Chapter 4. Chapter 5 describes the design of the adaptors, including the Domain and Toolchain models, as well as the resource mapping. Chapter 6 gives a summary of the implementation approaches and challanges faced during the development of

the adaptors. The testing and evaluation of the created adaptors are described in Chapter 7. The installation and usage guide can be found in Appendix A.

Chapter 2

Requirements Engineering

This chapter briefly covers one of the main parts of requirement engineering, which is requirements management, its fundamental concepts, and a basic overview of Jira Software and Requirements for Jira plugin. Information provided in this overview is important for understanding the requirements and needs of the adaptor.

2.1 Requirements Management

The aim of requirement engineering is the verification of the accomplishment of the project's goals and objectives. The process is divided into several successive stages, the most essential being analysis, documentation, tracing, and change control. The purpose of this process is to track the requirements and their status, identify inconsistencies and provide an overview of the project progress to concerned stakeholders [14].

One of the most important parts of requirement management is *traceability*. It is the ability to track and assess the state of the requirement and its changes during the development lifecycle, providing an audit trail, usually used for reporting to stakeholders. Traceability can be achieved by linking requirements to other artifacts – e.g., requirements, test cases, or build stories.

Requirements

Requirements are the foundation for determining the needs of system stakeholders and the system itself. They represent a condition or capability that must be met by a system or product [18].

They can be divided into three main categories, *functional*, *non-functional* and *safety* requirements. Functional requirements describe the system behavior or product features (e.g., the system will send an email with a forgotten password prompt when requested by the user), while non-functional requirements usually specify the system's performance or product properties (e.g., a task has to be completed under 200 ms). Safety requirements are defined for the purpose of risk reduction and are usually specified by the industry standards and regulations (e.g., the autonomous vehicle should not accelerate if the distance to the object in front of it is less than 1 m).

2.2 Requirement Management Systems

Requirement Management System is a software tool used for the management of project requirements during the development lifecycle. Currently, there are several different requirement management systems developed by competing companies available on the market. Some of the most popular ones are IBM Doors and Jira Software with Requirements for Jira plugin. A high-level overview of these options is provided in the following sections, as they are important in the context of this thesis.

IBM Doors

IBM Doors [17] is a requirement management system developed by IBM [16]. It is a complex tool designed to handle large projects with many requirements and stakeholders. The tool is commonly used in the aerospace, defense, and automotive industries. It provides a wide set of features to aid in the requirement management processes, such as requirement traceability, collaboration, and ease of integration with other IBM tools. It is also designed in a way to help companies comply with industry standards and regulations for requirement management and project development. The most recent version called *IBM Doors NG* also provides all of the previously mentioned features through the web user interface, REST API as well as native OSLC interface.

Jira Software and Requirements for Jira

Jira Software [19] is an issue-tracking and project management tool developed by Atlassian [2]. Compared to IBM Doors, it provides a more user-friendly interface and is more flexible, as its functionality spans across many different areas, such as project management, test management, and bug tracking. It also allows the end customers to extend the capabilities of their Jira instance by adding plugins developed by Atlassian or third-party developers, augmenting the original functionality, or adding completely new features. The contents of Jira and its functionality are accessible through the web user interface or REST API [22] with BASIC or OAuth2 authentication, further described in chapter 4. It does not provide a native OSLC interface, but the support for OSLC can be added by installing a third-party plugin (e.g., OSLC Connector for Jira [31] for Change Management Specification).

Requirements for Jira (R4J) [52] is a native Jira plugin extending the capabilities of Jira by adding support for requirements management by making use of Jira issues. It provides the ability to manage requirements by creating a folder structure, enabling importing and exporting in the reqIF format, enabling traceability, by utilizing the Jira link functionality, and adding the option to export these links into a comprehensive traceability matrix. All of these features are available both from the Jira web user interface as well as from the R4J REST API [53]. Because the requirement management system provided by R4J is a part of Jira, it allows for better traceability, as the requirements can not only be linked to other requirements but also to other Jira issue types, such as bugs, stories, test cases, and many more.

2.3 Requirements Interchange Format

Requirements Interchange Format (ReqIF) [57] is an XML-based format used for transferring and sharing requirements between requirement management systems or other tools.

The format defines a standardized way to describe requirements, their attributes and properties, relations between requirements, and a hierarchical structure, in which the requirements are contained. Each object, type, attribute, and relation contains a unique identifier by which it can be referenced from other objects.

The document is divided into two parts, `THE-HEADER`, containing metadata information about the requirement collection, and `CORE-CONTENT`, which is then further split into five sections, each containing different information about the requirements:

- `DATATYPES` – definitions of datatypes used in the document
- `SPEC-TYPES` – definitions of types of requirements, its attributes and relations, the data type of the attribute is defined by a reference to a definition in `DATATYPES`
- `SPEC-OBJECTS` – a collection of requirements and its properties, described in a format specified in `SPEC-TYPES`
- `SPEC-RELATIONS` – a collection of relations between requirements, described in a format specified in `SPEC-TYPES`
- `SPECIFICATIONS` – requirements hierarchical tree structure

An extract of a small sample file to illustrate the structure of a ReqIF document can be seen in Listing 2.1.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<REQ-IF xmlns="http://www.omg.org/spec/ReqIF/20110401/reqif.xsd">
  <THE-HEADER>
    <REQ-IF-HEADER IDENTIFIER="654d6119-6d4f-4ab9-b43f-39e4360353ff">
      <CREATION-TIME>2023-04-16T17:15:41.445316</CREATION-TIME>
      <TITLE>Sample title</TITLE>
    </REQ-IF-HEADER>
  </THE-HEADER>
  <CORE-CONTENT>
    <REQ-IF-CONTENT>
      <DATATYPES>
        <DATATYPE-DEFINITION-STRING IDENTIFIER="Text"/>
      </DATATYPES>
      <SPEC-TYPES>
        <SPEC-OBJECT-TYPE IDENTIFIER="FUNC-REQ" LONG-NAME="Requirement">
          <SPEC-ATTRIBUTES>
            <ATTRIBUTE-DEFINITION-STRING IDENTIFIER="FUNC-REQ-TXT"
             LONG-NAME="Description">
              <TYPE>
                <DATATYPE-DEFINITION-STRING-REF>
                  Text
                </DATATYPE-DEFINITION-STRING-REF>
              </TYPE>
            </ATTRIBUTE-DEFINITION-STRING>
          </SPEC-ATTRIBUTES>
        </SPEC-OBJECT-TYPE>
      </SPEC-TYPES>
      <SPEC-OBJECTS>
        <SPEC-OBJECT IDENTIFIER="TEST-ID-1" LONG-NAME="Test REQ 1">
          <VALUES>
            <ATTRIBUTE-VALUE-STRING THE-VALUE="Test REQ description">
              <DEFINITION>
                <ATTRIBUTE-DEFINITION-STRING-REF>
                  FUNC-REQ-TXT
                </ATTRIBUTE-DEFINITION-STRING-REF>
              </DEFINITION>
            </ATTRIBUTE-VALUE-STRING>
          </VALUES>
          <TYPE>
            <SPEC-OBJECT-TYPE-REF>FUNC-REQ</SPEC-OBJECT-TYPE-REF>
          </TYPE>
        </SPEC-OBJECT>
      </SPEC-OBJECTS>
      <SPEC-RELATIONS />
    </REQ-IF-CONTENT>
  </CORE-CONTENT>
</REQ-IF>
```

Listing 2.1: A simple ReqIF file containing one requirement.

Chapter 3

OSLC – Open Services for Lifecycle Collaboration

This chapter provides a brief overview of the OSLC (Open Services for Lifecycle Collaboration) [30], its fundamental technologies, and the Core and Requirement Management specifications, which are used in this thesis.

3.1 Overview

OSLC [30] is an OASIS Open Project [28] responsible for developing a set of open specifications, which are used for easier integration of software tools. A more detailed overview of the OSLC project can be found in the *OSLC Primer* [42].

The initiation of the OSLC project was driven by the increasing number of software tools, which are used in the software development lifecycle. These tools are usually developed by different organizations, which leads to the problem of difficult tool integration. In the past, this was solved by developing specific translators and adaptors for each tool, which was a time-consuming and expensive process. OSLC was created to solve this problem, by creating a set of open specifications to integrate the resources managed by the software tools into the web of data.

OSLC offers two different methods of data integration – *Linking data via HTTP* and *Linking Data via HTML User Interface* [41].

Linking data via HTTP

Linking of the data via HTTP is based on OSLC-defined common tool protocol for accessing, creating, updating, and deleting resources. The protocol is based on internet technologies and standards, such as REST, RDF, and Linked Data, described in section 3.2. It allows any other tool, that implements the same specification, to access any of the managed resources. Linking of the data is done by referencing resources by HTTP URIs in the representations of other data.

Linking Data via HTML User Interface

OSLC protocol can be used to link data via HTML user interface as well, making use of the REST *Code on demand* optional constraint. This allows the client to access and display fragments of an existing user interface, provided by the tool, without the need to implement

the user interface itself. This delegated user interface then enables the client to access the resources managed by the tool.

3.2 Fundamental Technologies

OSLC is built on top of several fundamental technologies. This section lists these technologies and introduces briefly each of them. Figure 3.2 shows how these technologies are used together to create the base for the OSLC architecture.

REST

REST (REpresentational State Transfer) [12] is a web software architecture style, describing a set of constraints and properties, which should be followed in communication between computer systems, most commonly between client and server. In REST architecture, the server is responsible for exposing a common interface, which allows the client to access resources by using standard HTTP methods. Each resource the server provides has a unique identifier, that allows the resource to be identified unambiguously and completely. When requested, the server responds with a representation of the resource. The client can then use this resource to create new resources and update or delete existing resources. The communication between the server and the client is stateless, meaning every request the server receives can be fully understood in isolation, without the context of previous requests. The most common HTTP methods used in REST are `POST`, `GET`, `PUT`, and `DELETE`, which correspond to the CRUD operations (Create, Read, Update, and Delete). REST also defines one optional constraint – *Code on demand*, which allows the server to send executable code to the client, extending the functionality of the client.

RDF

RDF (Resource Description Framework) [58] is a W3C (World Wide Web Consortium) [69] standard for representing data on the web. It describes resources in the form of a directed graph, where information about each element is represented as a triplet – example of a single triplet can be seen in Figure 3.1. Triplets are statements about the resource composed of subject, predicate, and object. The subject describes the resource, and the predicate specifies its properties and relationships, between the subject and the object. Most widely used RDF serialization formats are Turtle [66], RDF/XML, and RDF/JSON.

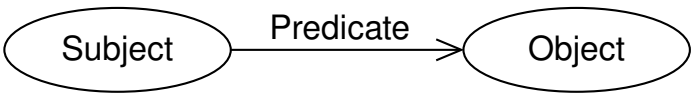

Figure 3.1: RDF Graph Triplet (source: [54])

Linked Data

Linked Data [24] are structured data containing references to other data. This enables computers to query and interpret the data, allowing the internet to become one big database. The main principles of Linked Data are [25]:

1. URIs [67] are used as names to identify things

2. People can lookup things using HTTP URIs

3. Information returned as a result of the search are provided in an open standard format (for example RDF)

4. Returned information contain more URIs, enabling discovery of other things

3.3 OSLC Specifications

OSLC defines a set of open specifications for integrating software tools. OSLC consists of multiple working groups, which are each responsible for the development of a specific specification. There are two types of specifications – *Core* and *Domain*. The Core specification provides a basis for the Domain specifications, which are then focused on a specific field, for example, Requirement Management, Change Management, Configuration Management, etc. This section provides a summary of the OSLC Core and Requirement Management specifications, which are used in this thesis.

3.3.1 OSLC Core Specification

At the time of writing this thesis, the current version of OSLC Core Specification [37] is 3.0. The OSLC Core Specification defines a set of common principles, capabilities, and restrictions, which should be common across all OSLC Domain Specifications. Specific OSLC Domain Specification will then describe which of these capabilities are required or optional for conformance with the specification. It also introduces several resource types and properties with the namespace `http://open-services.net/ns/core#` and prefix *oslc*. In the following sections, a basic overview of the core concepts is provided.

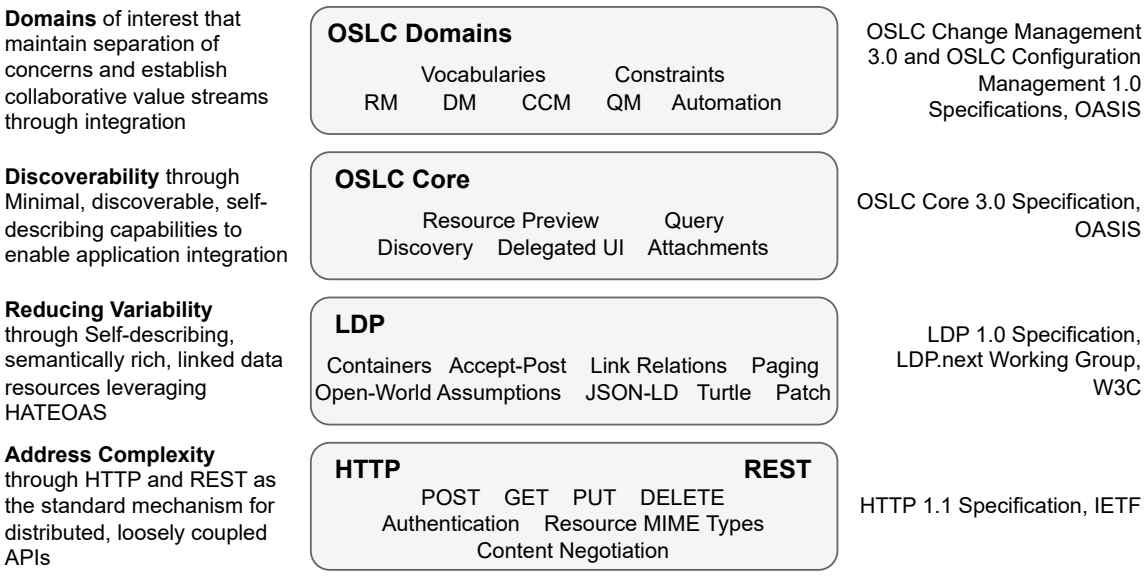

Figure 3.2: OSLC Core 3.0 architecture (source: [37], remade)

Resource Shape

OSLC works with resources, which are uniquely identified by a URI [67], and are represented by RDF triples. Resource Shape [36] is a resource of type `oslc:ResourceShape`, that describes the contents and constraints of other resources. Figure 3.3 shows a simple oriented graph representing a Resource Shape.

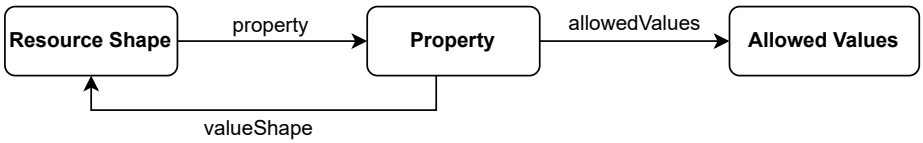

Figure 3.3: Resource Shape (source: [36], remade)

Each Resource Shape has a defined set of Properties, of type `oslc:Property`, which specifies the value type and cardinality of the property. The value type of the property can be either a reference to another Resource Shape, or a basic data type. The cardinality of the Property specifies if the Property is required, optional, or can be present multiple times. OSLC Core Specification defines these basic data types: `XMLLiteral`, `boolean`, `dateTime`, `decimal`, `double`, `float`, `integer`, `string`, and `langString`.

Discovery

For the reasons of flexibility and to reduce coupling, the OSLC Core Specification does not specify unequivocally which capabilities the server has to provide. Instead, it offers a mechanism for the incremental discovery of services and capabilities the target server has implemented. Figure 3.4 shows a diagram of the OSLC Discovery mechanism.

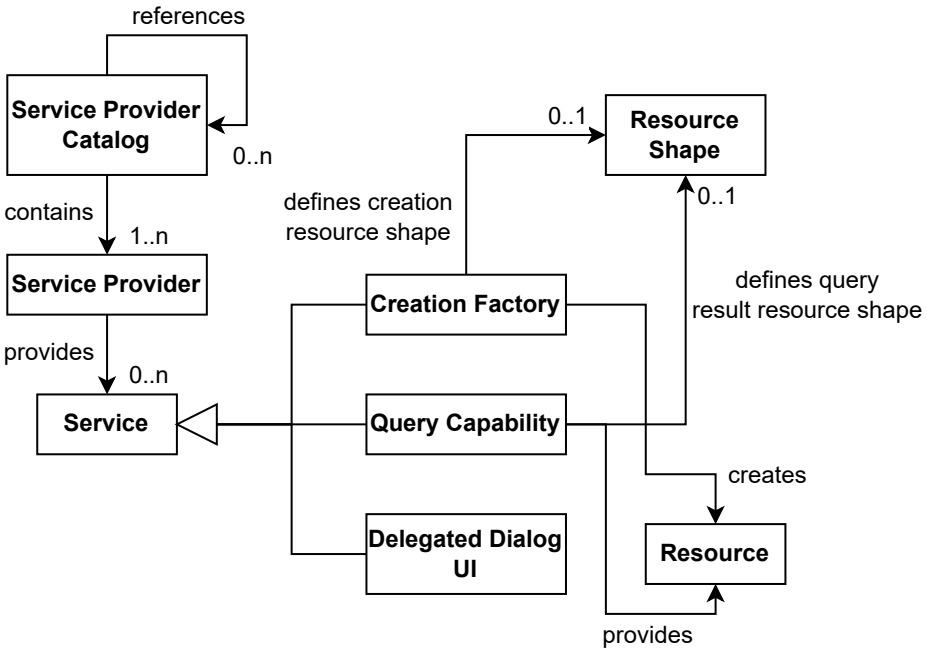

Figure 3.4: OSLC Discovery diagram (source: [33], remade)

The server always has to specify the starting point of discovery, which is the *Service Provider Catalog*. The Service Provider Catalog is a resource containing a list of available *Service Providers*, which then contains all of the available *Services*. From there, the client is able to find the URIs for *Creation Factories*, *Dialogs*, and *Query services*. Additional information about the Discovery mechanism can be found in the Discovery section of the OSLC Core Specification [33].

Basic Capabilities

OSLC Core Specification defines basic CRUD (Create, Read, Update, Delete) operations for resources. However, the decision about which of these operations are required or optional for which resource is specified in each of the OSLC Domain Specifications. Read, Update, and Delete operations are performed by their respective HTTP request method to the URI of the target resource. Create operation is performed by a `POST` operation to the Creation Factory URI for a specific resource.

Delegated UI

OSLC Core Specification introduces Delegated UI [32] for resource creation – *Creation Dialog*, and resource selection – *Selection Dialog*. Both of these dialogs are examples of linking of the data via HTML mentioned in 3.1. Dialogs are returned as a combination of HTML `iframe` and JavaScript code. The decision about which of these dialogs are required or optional is again left up to the OSLC Domain Specification.

Query Capability

As the OSLC server manages a large number of resources, it has to provide a way for clients to search and filter these resources. OSLC Core Specification specifies a mechanism for this, called OSLC Query [35]. OSLC Query allows clients to look up a set of resources by performing a `GET` or `POST` request on an `oslc:queryBase` URI. It offers two separate capabilities, a full-text search, identified by the `oslc.searchTerms` parameter, and a query search for resources containing specific properties and values, identified by `oslc.where` parameter. Each query search must consist of at least one property, comparison operator, and value. The result of the search is returned as a resource of type `oslc:QueryResult`, which contains a list of references to the resources found (example of `oslc:QueryResult` can be found seen in Listing 3.1).

Authentication and Error Responses

OSLC Core Specification provides guidance on how to handle authentication and error responses. Allowed authentication methods are Basic Authentication and OAuth. The details of the authentication process are further explained in Chapter 4.

All error responses should be returned as a resource of type `oslc:Error` [34], which contains a human-readable message and a machine-readable error code. This enables clients to handle errors in a generic way.

3.3.2 OSLC Requirement Management Specification

At the time of writing this thesis, the current version of OSLC Requirement Management Specification [47] is 2.1. The specification builds on top of the OSLC Core Specification

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:dcterms="http://purl.org/dc/terms/"
  xmlns:oslc_data="http://open-services.net/ns/servicemanagement/1.0/"
  xmlns:oslc="http://open-services.net/ns/core#"
  xmlns:oslc_rm="http://open-services.net/ns/rm#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:foaf="http://xmlns.com/foaf/0.1/">
  <oslc:ResponseInfo rdf:about="...">
    <oslc:totalCount rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
  >2</oslc:totalCount>
  </oslc:ResponseInfo>
  <rdf:Description rdf:about=".../queryRequirement">
    <rdfs:member>
      <oslc_rm:Requirement rdf:about="...">
        ...
      </oslc_rm:Requirement>
    </rdfs:member>
    <rdfs:member>
      <oslc_rm:Requirement rdf:about="...">
        ...
      </oslc_rm:Requirement>
    </rdfs:member>
  </rdf:Description>
</rdf:RDF>
```

Listing 3.1: OSLC Query Result example

and specifies which of the capabilities are required or optional for conformance with the specification. The main goal is to provide an extensive, but not restrictive, interface for requirement management systems and support a wide range of integration scenarios. One of the main requirements for an OSLC RM Server is the ability to accept and return resources in RDF/XML, XML, and JSON. It also introduces new resource types in the namespace of `http://open-services.net/ns/rm#` and with the prefix `oslc_rm`. These resource types are described in the following sections.

Requirement

`oslc_rm:Requirement` [43] is a resource shape used for describing a single requirement described in Section 2.1. Requirement Management Specification defines an extensive set of properties and constraints [46] for the requirement shape, but there are few that are especially important in the context of this work.

Each individual requirement should have a title, usually containing the name of the requirement, and a description, which consists of the actual statement of need. These two requisites are realized by the properties `dcterms:title` and `dcterms:description`.

Requirements should also be able to reference other requirements or requirement collections that are related to them. This is done by the properties `oslc_rm:decomposedBy` and `oslc_rm:decomposes`, the difference being the direction of the relationship.

Requirement Collection

`oslc_rm:RequirementCollection` [44] is a resource shape used for describing a collection of requirements, which constitute some statement of need. Requirement Management Specification again defines the constraints and properties for this resource shape [45], but they are nearly the same as for `oslc_rm:Requirement` resource shape.

3.4 Eclipse Lyo

Eclipse Lyo [9] is an open-source project hosted by the Eclipse Foundation [7] and developed by the OSLC community. It provides a Java SDK, as well as other utilities, to enable easier adoption of the OSLC technologies and better developer experience. The following sections give a concise summary of the key components of Eclipse Lyo, which are used to implement the OSLC adaptor in this thesis.

OSLC4J SDK

OSLC4J Software Development Kit (available at Maven Repository [27]) is a set of Java libraries used for building OSLC-compliant REST-based servers and clients. It includes support for common OSLC capabilities, resource shapes, service provider documents, and marshaling and unmarshaling[1] of resources to Java objects.

Lyo Designer

Lyo Designer [26] is an Eclipse plugin used for the graphical design of OSLC adaptors. It offers the capability to model the OSLC resources, their properties, and constraints, as well as the OSLC services. For separation of concerns, Lyo Designer provides three views for modeling different parts of the final adaptor:

- **Domain Specification View** – for modeling the OSLC resources, their properties, and relationships between them
- **Toolchain View** – for modeling the relationships between separate adaptors and resources, they consume and produce
- **Adapter Interface View** – for modeling the services and capabilities of a single adaptor

Lyo Designer also contains a utility called **Code Generator**, which is capable of generating code skeletons, compliant with OSLC, from the graphical models of the adaptors designed in Lyo Designer. The generated code is based on the OSLC4J SDK and can be used as a starting point for the implementation of the adaptor. Out of the box, the

[1] **Marshaling** is the process of transforming the representation of data from memory into the format used for transmission. It is used for passing the data of an Object to a remote server, with the *CodeBase* attached, specifying where the implementation of the object can be found.

generated code contains the definitions of the OSLC resources, their properties and constraints, definitions of the OSLC services, and code for marshaling and unmarshaling of the resources. The program logic (how the resources should be handled after unmarshaling) and implementation of the services are left to be done by the developer. The generated code contains designated places where the developer should add code, which provides the functionality for the adaptor. This allows the code to be regenerated, upon the changes to the model, without breaking or losing any of the underlying implementation and custom code.

Chapter 4

Authentication

Authentication is the process of verifying the identity of a person or user. It is used to restrict the server resources or functionality to only authorized users or specific groups of users. Several different authentication methods exist, each with its own advantages and disadvantages. This chapter gives a brief overview of BASIC and OAuth authentication methods, which are used in this thesis to secure the OSLC adaptors as required by the OSLC specification.

4.1 Basic Access Authentication

BASIC (Basic Access Authentication) was first defined as a part of HTTP 1.0 specification [15], but the standard has been since superseded and redefined as part of its own RFC [63]. As its name suggests, BASIC is the most fundamental method of verifying the identity of a client against a server. It utilizes the HTTP `Authorization` header and provides the server with the credentials of the client in the form of `Authorization: Basic <credentials>`. The credentials are provided as a string, encoded in base64 [62], containing the username and password joined by a colon.

The main advantage of using BASIC authentication is its ease of implementation. It is supported by most of the available frameworks and does not require any cryptographic operations. However, it is also the most vulnerable one, as the credentials are sent in plain text, which makes it susceptible to interception by a third-party and potential credentials theft. Most of these insecurities can be mitigated by using TLS to encrypt the communication between client and server, but it is still not a recommended method for authentication in production environments.

4.2 Open Authorization

OAuth (Open Authorization) is a standard for delegated authentication and authorization, which stemmed from the need to enable the end users to authenticate in third-party applications and services using their credentials from another service, which acts as the authorization server. Originally, this was done by sharing the credentials with the third-party application, which then used them to authenticate the user. This method was vulnerable to the same security issues as BASIC authentication, and on top of that, it also allowed the third-party application to access and read the user's credentials. OAuth was created to mitigate these problems and create a standardized way for services to offer authenti-

cation and authorization to third-party applications without the need to share the users' credentials between them. The standard was first introduced as OAuth 1.0 [64], which used asymmetric cryptography to encrypt and verify the credentials, but was later superseded by OAuth 2.0 [65], which is easier to implement, as it leaves the encryption and verification of the credentials origin to TLS protocol. The next section provides a brief overview of OAuth 2.0 and its authentication workflow.

4.2.1 OAuth 2.0

OAuth 2.0 is a standard published as a reaction to new use cases. It is not backward compatible with OAuth 1.0. The standard enforces that all of the communication between the client and the authorization server is done using HTTPS.

Several different roles are defined by the standard:

- **Resource owner** – entity capable of granting access to a protected resource
- **Resource server** – server containing the resources, capable of authorization using the access tokens
- **Client** – third party application, which is requesting access to resources on the resource server on behalf of the resource owner
- **Authorization server** – issues access tokens to resource owner after successful authentication and authorization

Code Grants

The standard also defines four authorization grants, ways for a client to obtain the access token. For simplicity, only the most common one, Authorization Code Grant, is described, as it is the one used in this thesis.

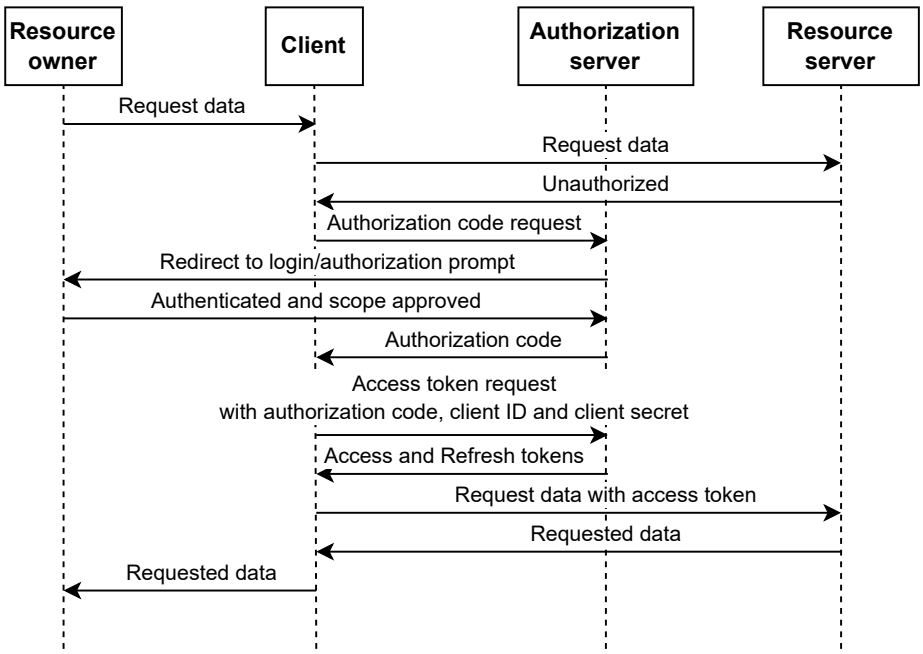

Figure 4.1: OAuth 2.0 Authorization Code Grant flow diagram

Authorization Code Grant is a two-step process to obtain the access token. The user is first redirected to the authorization server with a request to obtain an authorization code. The authorization server authenticates the user and asks him for approval to grant the client specified scope of access to resources on the resource server. After the approval, the user is redirected back to the client with the authorization code. The client then uses the code together with client ID and client secret to make a request to the authorization server to obtain the access token. The authorization server then verifies the authorization code and issues the access token. The whole process is depicted in Figure 4.1.

Proof Key for Code Exchange

PKCE (Proof Key for Code Exchange) [51] builds on top of Authorization Code Grant to further secure the process. It adds a secret called *Code Verifier*. Code Verifier is then to transformed into a value called *Code Challenge*, which is sent together with the authorization code request. Code Verifier is then sent together with the authorization code as a part of the access token request. The authorization server issues a new access token only if the received Code Verifier matches the one used to generate the Code challenge. The modified flow is depicted in Figure 4.2. This removes the risk of the authorization code being intercepted and used to generate the access token, as the authorization server will not create the access token without the Code Verifier, which is not part of the intercepted response.

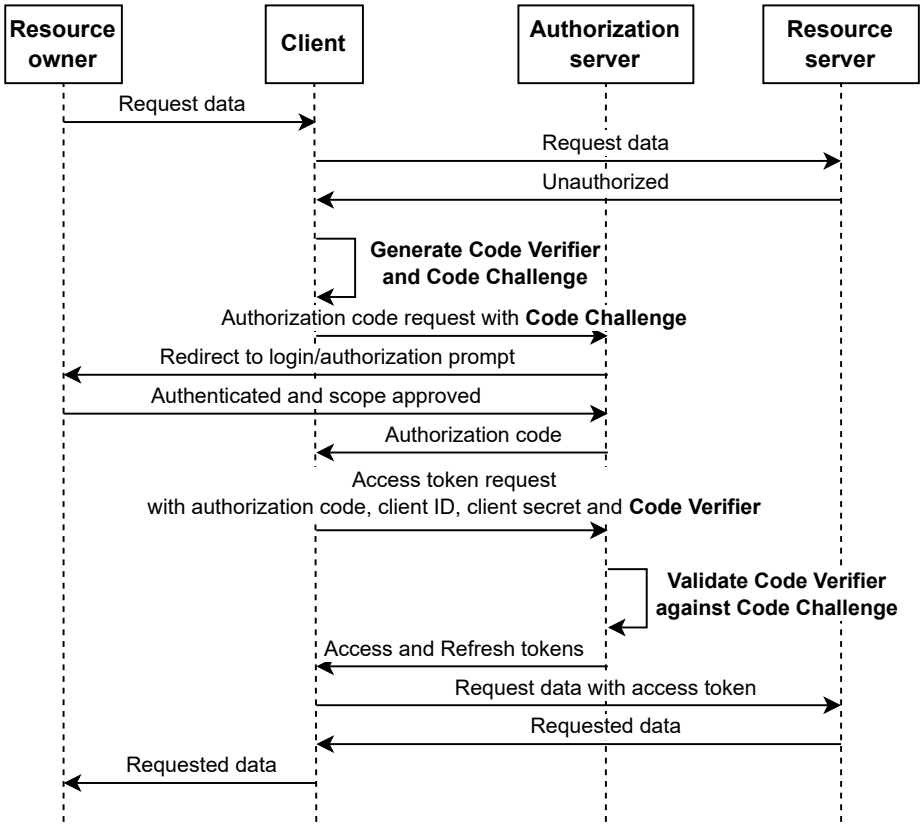

Figure 4.2: OAuth 2.0 Proof Key for Code Exchange flow diagram

Access and Refresh Tokens

After successful authorization against the authorization server, the client is issued an **access token**. The client then uses this token to access the resources available on the resource server by sending it with each HTTP request in the Authorization header. Access tokens can be either an opaque string, called a *bearer token*, or a *sender-constrained token*. Sender-constrained tokens can be used only to authorize requests sent by the same client to whom the token was issued. This is usually achieved by asymmetric cryptography of the token. To comply with the standard, the token cannot convey user identity or any other information about the user.

Together with the access token, the authorization server also issues a **refresh token**, which can be used to generate a new access token without user interaction with the authorization server. This allows the authorization server to issue access tokens with a shorter expiration time, reducing the impact of the token being intercepted and stolen.

Scopes

Compared to OAuth 1.0, OAuth 2.0 also introduces the concept of *OAuth Scopes*, allowing the authorization server to issue access tokens with various limitations on access to resources. The scopes are defined by the resource server. The requested scope is part of the authorization request to the authorization server and is bound to the access token generated as a result of the authorization. This can be used to limit the generated access token to read-only access, which is useful for third-party applications that do not need to modify the resources on the resource server.

Chapter 5

Adaptor Design

This chapter details the process of designing the Jira and R4J adaptors, as well as the decisions made during the process and the reasoning behind them. The chapter is divided into several sections, each detailing different steps of the design process.

5.1 Architecture Overview

There are two possible ways how to add the OSLC capabilities to a third-party web application – either by extending the application via a plugin or by creating a separate self-contained application that acts as a middle-man between the third-party application and the user. Both of these approaches can be used to create the adaptor for Jira and R4J, as Jira allows the creation of third-party plugins and exposes its data through the REST API. At the start of the design process, a decision was made to keep the adaptor as generic as possible, to allow the option to turn it into a generic adaptor for requirement management specification in the future. This would allow its use with other third-party requirement management applications by utilizing the strategy design pattern [5] to switch the data-access layer of the adaptor. With this in mind, the second approach, to create a standalone web application, was chosen, as there is no way to create a plugin that would be compatible with every possible third-party application.

After careful consideration of the capabilities and data exposed by both Jira and R4J, it was determined that the best option would be to split the adaptor into two separate adaptors, one for Jira and the second for R4J. This is in line with the decision to keep the adaptor as generic as possible. The Jira adaptor would be responsible for satisfying the requirements of the *OSLC Core* and *Requirement Management Specification*. The R4J adaptor would add some additional functionality provided by R4J while not putting any restrictions on what functionality has to be provided by any other third-party requirement management applications in order to be compatible with the generic adaptor. These adaptors can then be used together to provide the full functionality, or users can opt to use only the Jira adaptor if they do not need the additional functionality or do not have an active subscription to the R4J plugin. The Jira adaptor still retains compliance with the OSLC Core and Requirement Management Specification while used alone.

5.2 Tools Selection

One of the initial steps of the design process was the selection of the language, libraries, and tooling that would be used to implement the adaptors. At the time the design process started, there were only two real language options available for consideration, being `C#` and Java.

The *OSLC4Net* [49] library for `C#`, developed by the OSLC team, was in an unusable state at the time, as it has been overlooked and not maintained for a long time. It has recently started getting some attention at the time of writing this thesis, so if the library is brought up to date and additional tooling is developed, it might become a viable option in the future.

Because of the state of OSLC4Net, the only viable option left was to use Java with the well-maintained OSLC4J SDK, together with the Eclipse Lyo tooling for the development of OSLC applications. The code generator included in Eclipse Lyo generates the code skeletons for the adaptors as Eclipse Jetty servers [8] equipped with Apache Maven [1] for dependency management and build automation.

5.3 Adaptors Modeling

After tooling selection, the next step in the design process was to model the adaptors themself. The modeling was done using the Eclipse Lyo modeling tool, which also provides the ability to generate code skeletons from the created models. Most of the modeling process is based on the tutorials [38] [48] [10] and examples [39] [11] provided by the OSLC community, as well as adaptor models for Unite [68]. The modeling of the adaptors can be separated into two steps, domain modeling – modeling the resources accepted and exposed by the adaptor, and toolchain modeling – modeling the functionality and capabilities of the adaptor, both of which are described in the following sections.

5.3.1 Domain Modeling

The domain models were created based on the OSLC Core and Requirement Management specifications, with the data exposed by Jira and R4J APIs taken into consideration. The requirements and restrictions placed on the `oslc` and `oslc_rm` are defined in the Requirement Management specification using the `MUST`, `MUST NOT`, `SHOULD`, `SHOULD NOT` and `MAY` keywords. The main goal was to create models compliant with the standards while taking into account their extensibility in the future and without creating any limitations by coupling them with Jira or R4J-specific resources. This lead to the separation into `oslc`, `oslc_rm`, `jira` and `jira_r4j` models. The `oslc` and `oslc_rm` models were modeled using the models provided by Lyo project [40] as a base.

OSLC Core Domain

The OSLC Core domain model is specified by the OSLC Core specification described in 3.3.1. As every OSLC-compliant adaptor has to implement the OSLC Core specification, the domain model provides the base resources of the OSLC Core vocabulary as well as other basic namespaces used for the definition of other resources – namely RDF [59], RDFS [60], Dublin Core [6] and FOAF [13]. The OSLC Core domain model can be seen in Figures 5.1 and 5.2.

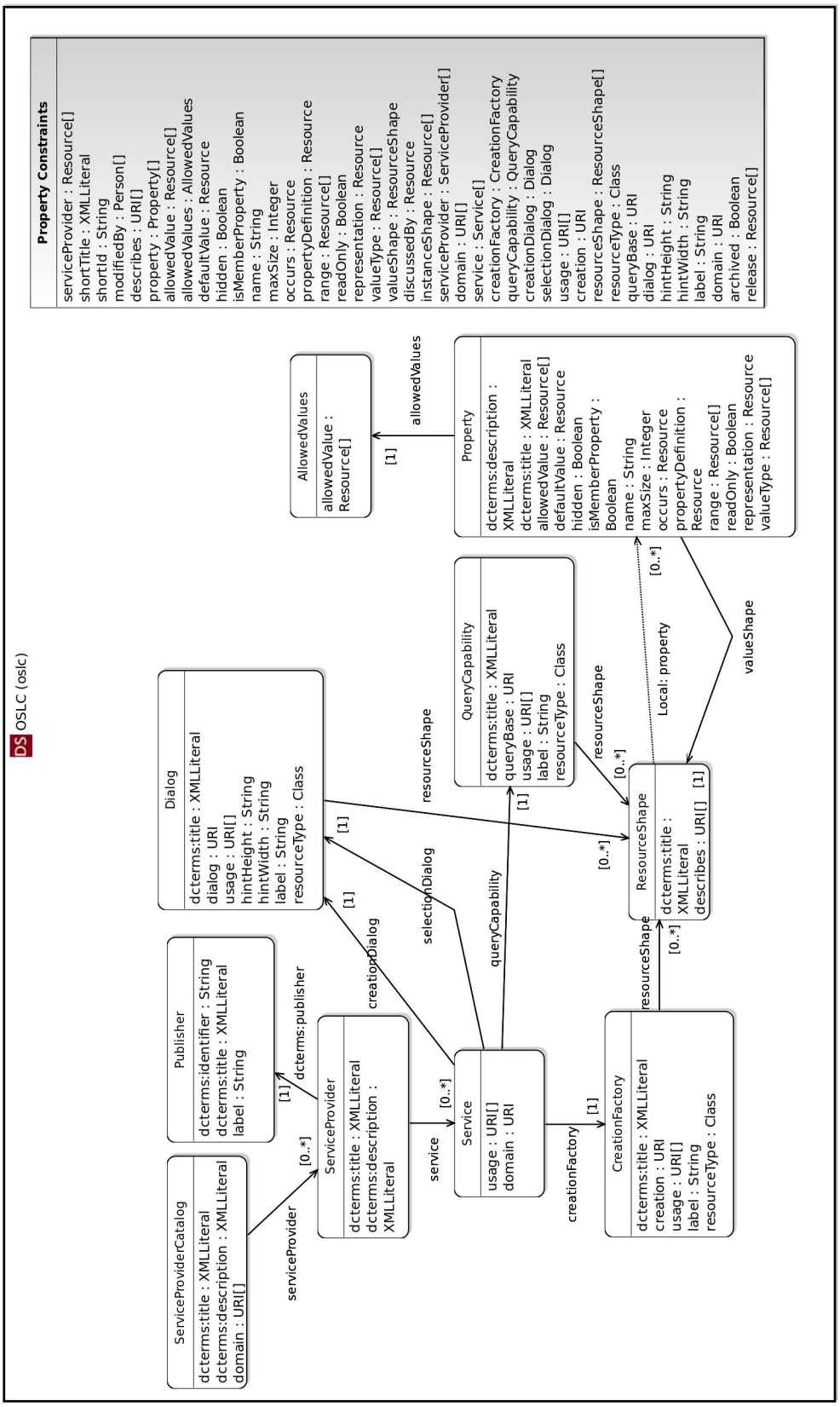

Figure 5.1: OSLC Core Domain Model Diagram *(Exported from Lyo modeling tool)*

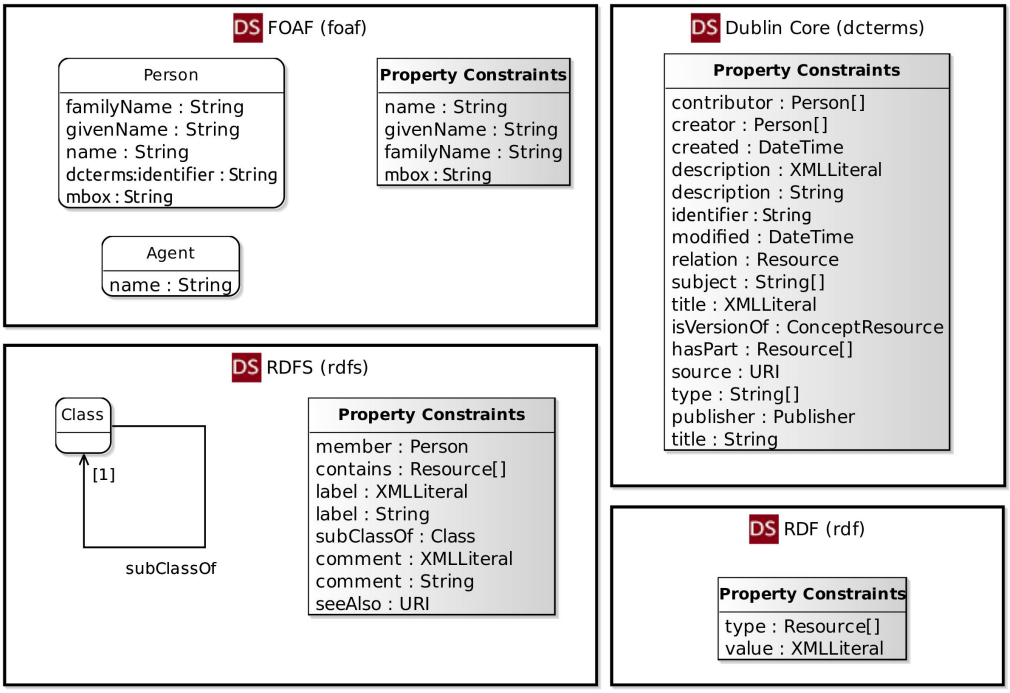

Figure 5.2: OSLC Core Model Diagram *(Exported from Lyo modeling tool)*

OSLC Requirement Management Domain

The OSLC Requirement Management domain model defines the `oslc_rm:Requirement` and `oslc_rm:RequirementCollection` resources based on the models from the Lyo project. Properties that are not required by the Requirement Management specification and are not needed in this thesis have been removed, and the representations have been extended by the properties defined in `jira` namespace. The graphical representation can be seen in Figure 5.3.

OSLC Jira and R4J Domains

The `jira` and `jira_r4j` namespaces define properties and resources that represent resources and properties native to Jira and R4J. The `jira_r4j:Folder` has been defined in a way where it references `oslc_rm:Requirement` and `oslc_rm:RequirementCollection`, instead of the other way around, to allow the use of the Jira adaptor independently of the R4J adaptor. The models can be seen in Figure 5.3.

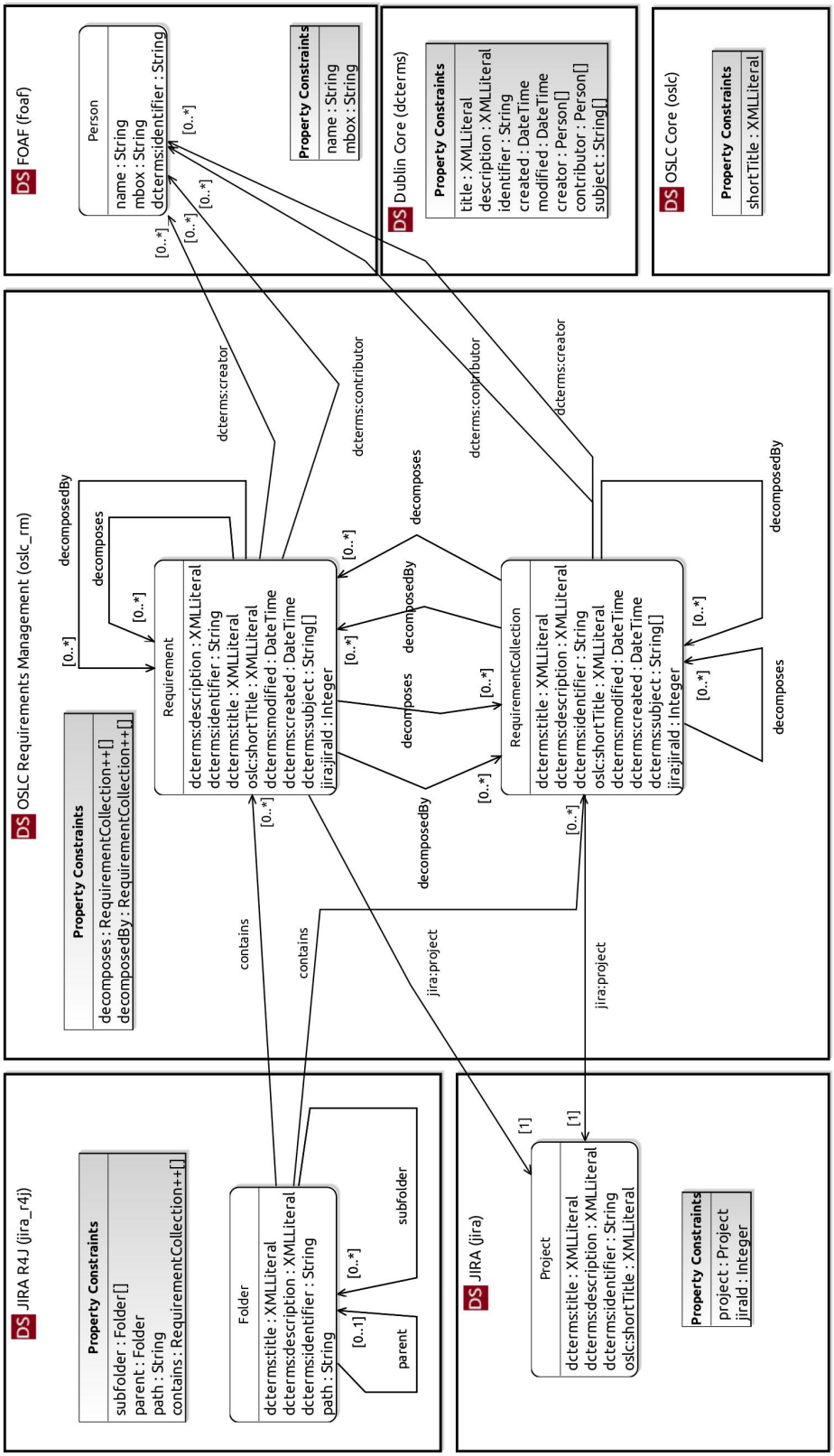

Figure 5.3: OSLC Requirement Management Domain Model Diagram *(Exported from Lyo modeling tool)*

5.3.2 Toolchain Modeling

As discussed before, the adaptor has been split into R4J and Jira adaptors. The adaptor interfaces can be seen in Figure 5.4. The Jira adaptor manages the `oslc_rm` and `jira` resources and properties, while the R4J adaptor is dependent on these resources and produces `jira_r4j` resources.

Both adaptors have been modeled according to the OSLC Core Discovery principles discussed in Section 3.3.1 and thus contain *Service Provider Catalogs*, *Service Providers* and *Services* for each managed resource. Eclipse Lyo also provides the option to add *Authentication* to the adaptor models, which results in the generation of the authentication and authorization code skeletons for Basic and OAuth methods during the code generation process. This option has been added for both adaptors.

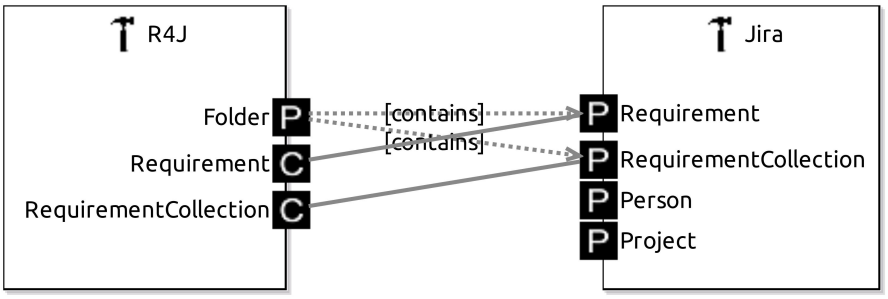

Figure 5.4: Adaptor Interfaces Diagram (`P` – *produces*, `C` – *consumes*)

Jira Adaptor

The Jira adaptor provides the `oslc_rm:Requirement`, `oslc_rm:RequirementCollection`, `foaf:Person` and `jira:Project` resources.

It exposes only limited functionality for the `foaf:Person` and `jira:Project` resources, allowing only read-only access to the data through the `GET` endpoint and *Selection Dialog*, as these resources are not meant to be created or updated using the adaptor.

In accordance with the Requirement Management specification, the Jira adaptor provides full *CRUD* functionality (as described in section 3.3.1) for the `oslc_rm:Requirement` and `oslc_rm:RequirementCollection` resources, as well as *Query capability*, *Creation Dialog* and *Selection Dialog*.

A graphical representation of the Jira adaptor functionality and capabilities can be seen in Figure 5.5.

R4J Adaptor

The R4J adaptor manages the `jira_r4j:Folder` resource, which represents a folder from the R4J tree folder structure. To enable users to use the R4J folder structure to its full extent, the adaptor provides full *CRUD* functionality for the resource. To further enable users and enhance the usability of the adaptor, the *Query Capability*, *Creation Dialog*, and *Selection Dialog* have been added to the adaptor.

A graphical overview of the R4J adaptor functionality and capabilities can be seen in Figure 5.6.

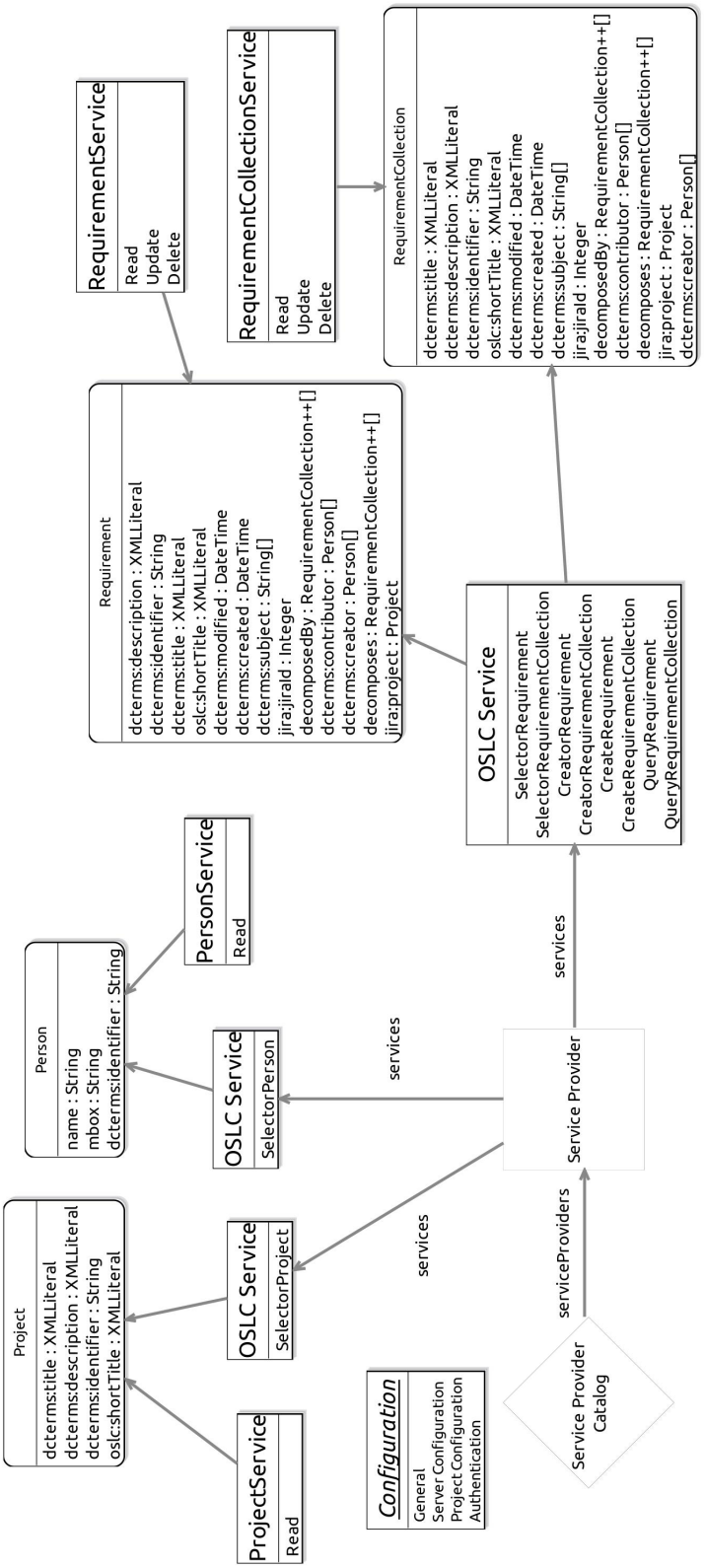

Figure 5.5: Jira Adaptor Functionality Diagram *(Exported from Lyo modeling tool)*

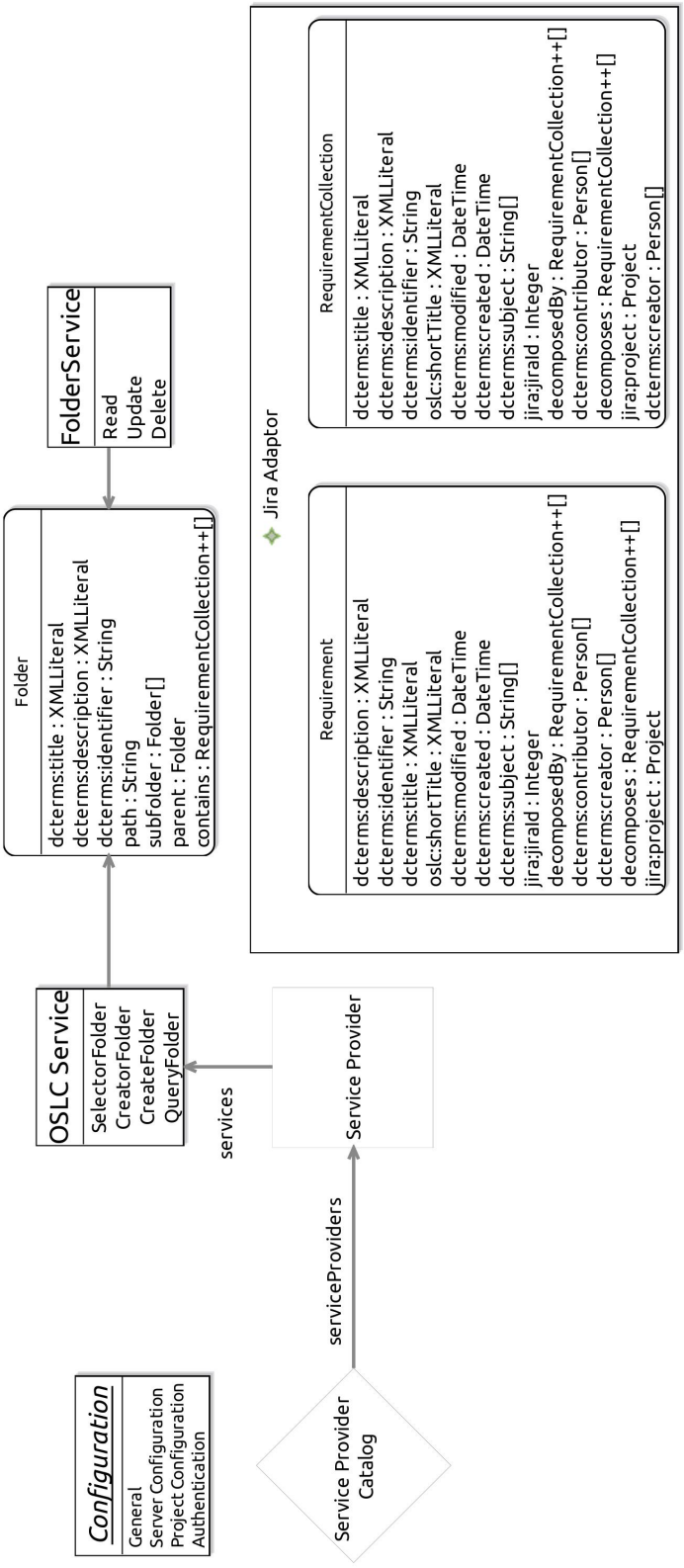

Figure 5.6: R4J Adaptor Functionality Diagram *(Exported from Lyo modeling tool)*

5.4 Mapping OSLC resources to Jira and R4J

Three important aspects have to be considered when designing the mapping of OSLC resources to their Jira and R4J counterparts. This section details these aspects and the design decisions made as a result.

Generic nature of the adaptor

The first is the generic nature of the adaptor for Requirement Management specification, taking into consideration the fact that requirement management applications, different from Jira, might not have exactly the same data structure and thus the mapping has to be flexible enough to allow for the use of the adaptor with other applications in the future. This had the biggest impact on the design of the unique identifier of the `oslc_rm:Requirement` and `oslc_rm:RequirementCollection` resources. Jira provides two ways to uniquely identify the issues:

- `id` – a unique immutable number assigned to each issue upon creation
- `key` – a unique immutable string identified assigned to each issue upon creation, consisting of a project key and a number, which identifies the issue within the project

Neither of these options is ideal, as the `id` would require all the other requirement management applications to identify resources strictly by a number and the `key` would require the existence of the concept of a project in every requirement management application, both of which limit the generic usability of the adaptor. Based on these observations, a new unique string-based identifier has been added and mapped to the `dcterms:identifier` property of the `oslc_rm:Requirement` and `oslc_rm:RequirementCollection` resources. As the identifier is string based, it leaves the decision of the format of the identifier up to the user or underlying application. To enable the users to query the resources by the Jira native identifiers as well, they have been mapped to the `jira:jiraId` and `oslc:shortTitle`, respectively.

Infrastucture limitations

The second aspect that has been considered are the limitations that can be placed on the Jira instances by infrastructure teams. In bigger companies, this concerns Honeywell as well, the Jira is usually used by a variety of teams, each having different use cases. To ensure the needs of each of these teams are satisfied without introducing any issue schema-breaking changes, the extent of the Jira instance configuration is limited and managed by a separate infrastructure team. The adaptor must be configurable to allow for the use with Jira instances with different configurations while still providing the same functionality.

The main configuration points of Jira, which are important in the context of this thesis, are:

- `issue_type` – the type of the issue, which determines the fields available for the issue
- `issue_link` – the type of the link between two issues
- `field` – the field representing a property of the issue

Ideally, two new `issue_types` should be added to the Jira configuration to represent the `oslc_rm:Requirement` and `oslc_rm:requirementCollection` resources, however, it is possible to use any two of the existing `issue_types` native to the Jira base configuration as well. The same applies to the `oslc_rm:decomposes` and `oslc_rm:decomposedBy` properties, which are mapped to a singular `issue_link`, with each representing a different direction of the link.

The last problem point of the configuration is the newly added string-based identifier `dcterms:identifier` property. The adaptor allows two ways to store the identifier in the Jira issue:

1. `labels` – the identifier is stored in the `labels` field together with the values of the property `dcterms:subject`
2. `custom_field` – the identifier is stored in a custom `field`

Limitations of the Jira and R4J

The last, but not least, aspect that impacted how the mapping of the resources was designed, is the limited modifiability of some of the Jira and R4J resources and how these resources are provided by the Jira and R4J APIs.

There is no way to add a custom field to the representation of the Jira project, resulting in the inability to use anything except the numeric project `id` as the project's unique identifier. A similar issue was identified with the identifier for the `jira_r4j:Folder` resource, which is identified uniquely by the numeric `id`, only in the context of the project it belongs to. To comply with the OSLC Core specification, each resource has to be uniquely identified within the context of the whole adaptor. As a solution to this problem, the `jira_r4j:Folder` resource is uniquely identified by the combination of the folder `id` and the project `key` in the form of `<project_key>-<folder_id>`. This is still in line with the idea of the generic nature of the adaptor, as the project and folder identifiers are defined as part of the `jira` an `jira_r4j` namespaces and are not part of the generic `oslc_rm` domain model.

Chapter 6

Implementation

This chapter details the implementation of the adaptors, including the challenges and problems faced during the implementation and the solutions to these problems.

6.1 Code Architecture

Code Generator from the Lyo project was used to generate the code skeletons for the adaptors based on the models defined in Section 5.3. The code skeletons were generated as two separate Java jetty [8] projects based on the Maven [1] framework. These generated projects contain the definitions of each service the adaptor was modeled to provide, definitions of the modeled resources, and base implementation of the BASIC and OAuth authentication.

A third project was created to contain the shared code and functionality between both adaptors. This project contains `Helper` and `Builder` classes, which provide `static` functions to abstract away some of the data manipulation and condition checking. The `shared` project also implements `ConfigurationProvider` and `SessionProvider` as *Singletons* [4] and provide both adaptors with an easy way to access the configuration and session data.

To unify the way both adaptors access and modify the resources provided by the Jira and R4J APIs, the *Facade* [3] design pattern was used. A facade was created for each resource type, which provides the adaptors with a unified interface to access and modify the resources of the given type. All of these facades extend the `BaseFacade` implemented in the `shared` project, which enables the facades to access the Jira and R4J Clients.

6.2 Configuration and Build Scripts

To support different configurations needed because of the infrastructure limitations, as discussed in Section 5.4, the adaptors have to be configurable using an external configuration file. The configuration file is a JSON file specifying the different configuration points of the adaptor, such as the `issueType` for the representation of resources, the `issueLinkType` that should be used for relations, the `Url` of the Jira instance, enabled authentication methods and much more.

Build scripts for Linux-based systems were created to simplify the build and start-up process of the adaptors. These scripts are written in Bash and utilize the Maven console functionality to build and run the adaptors. The scripts first build the contents of the `shared` project and then both of the adaptors, as they are dependent on the `shared` project. During the start-up, the scripts validate the presence of configuration files as well as the

response of the started adaptors. If the adaptors are not started successfully, the scripts terminate with an error message.

6.3 Jira API Java SDK

Atlassian provides developers with a Java SDK for the Jira API [20]. This SDK contains the implementation of several asynchronous Jira API clients, class representation for Jira resources, response parsers, and base support for authentication and authorization. To support all of the functionality required by the adaptors, some clients had to be extended, and some new clients had to be implemented utilizing the existing underlying infrastructure.

The authorization of the requests sent by the clients is handled by the implementations of the `AuthenticationHandler` interface. The implementation of BASIC authentication is provided by the SDK. However, an implementation for token-based authorization had to be implemented to support OAuth2 authentication. The implemented solution is further described in the following section.

6.4 Authentication

The *Code Generator* from the Lyo project generates the base implementation of the BASIC authentication and OAuth1.a[1] authorization. However, this implementation is not usable in the context of the Jira and R4J adaptors, as the generated code treats the adaptors as *authentication authorities*. This means it expects the adaptors to validate the users' credentials and provide the users with authorization codes and access tokens. The authentication and authorization have to be handled by the Jira instance, which is the authentication authority in this case.

It was decided to use the OAuth2 authentication instead of *OAuth1.a* as it is a more modern and secure authentication method while being easier to implement and use. A decision was made to use the *OAuth2 Authorization Code Grant*, summarized in Section 4.2.1, with the requests for authorization codes and access tokens being handled by the Jira instance. This is still in line with the OSLC Core specification, as it does not require the adaptors to support or handle the authentication. To support the discoverability of the endpoints for the authorization and token, their URLs were added to the `ServiceProvider` resources of the adaptors.

The authorization of the requests to the adaptors, either by BASIC credentials or OAuth2 access tokens, is handled by forwarding the contents of `Authorization` header from the request to the Jira and R4J clients.

6.5 Creation and Update Capability

This section details how the creation and update flows have been implemented for the `Requirement`, `RequirementCollection`, and `Folder` resources. As both `Requirement` and `RequirementCollection` resources are saved as Jira issues with different `issueType`, they share the underlying generic implementation for `Issue` creation and update.

[1] **OAuth1.a** [29] is a revised specification of the OAuth1.0 addressing a session fixation attack

Issue Resource

The creation of the Jira issues is implemented as a sequence of validation and creation steps. The order of these steps is as follows:

1. Validate that the issue contains all required properties for its creation
2. Validate that all issues linked by `decomposedBy` exist
3. Validate that all issues linked by `decomposes` exist
4. Validate that the specified `project` exists
5. Validate that the specified `issueType` exists in the context of the project (`issueType` to be used for `Requirement` and `RequirementCollection` resources is specified by the configuration)
6. Validate that no issue with the same identifier exists
7. Validate that the fields for storing `subject` and `identifier` properties exists for the specified `issueType` (the fields for `subject` and `identifier` are specified by the configuration)
8. Create issue with the specified properties
9. Update the created issue with the `decomposedBy` and `decomposes` links

The update functionality is implemented in a similar way to the create functionality, the main difference being that the update validates if the issue with the specified identifier exists and if it is of the correct `issueType`. To correctly update the links between the issues, the existing links created by the adaptor are removed before the new links from the updated resource are created.

Folder Resource

The creation of the R4J folder resources is implemented as a chain of validation and creation steps. The order of these steps is as follows:

1. Validate that the folder contains all required properties for its creation
2. Validate that the specified `project` exists
3. Validate that the specified `parentFolder` exists
4. Validate that the `parentFolder` does not contain a folder with the same name as the one being created
5. Validate that all issues linked by `contains` exist while validating the `issueType` of the issues as well
6. Create folder
7. Update the created folder with the `contains` links

The update is done in a similar way to the create functionality, except the update validates if the folder with the specified identifier exists. To assert that the folder contains only the issues specified in the update, all of the issues currently linked to the folder are removed, and then new links are created.

6.6 Query Capability

The skeletons and endpoints for the query capability were generated by the Code Generator. However, the generated implementation was missing `oslc.searchTerms` parameters, which are required by the Requirement Management Specification. The `oslc.searchTerms` parameter was added to each of the generated implementations, thus enabling a full-text search of the resources.

As for the `oslc.where` parameter, there was no library for parsing the `oslc.where` query available at the time of writing this thesis. Therefore a simplified implementation of the `oslc.where` query parser was created, which supports the basic query capabilities for common use cases, as the full implementation of the query parser is well beyond the scope of this thesis because of the complexity of the *OSLC Query Language*.

The following sections describe how the parsed query was translated for the Jira and R4J APIs.

Jira Resources

For querying the `oslc_rm:Requirement` and `oslc_rm:RequirementCollection` resources, which are saved as Jira issues, the JQL (Jira Query Language) [21] was used. The JQL is SQL-like query language, which can be passed as a parameter to the search issues endpoint of the Jira API in the `jql` parameter.

To map the parsed query to the JQL, an `IssueTranslator` was implemented utilizing the `JiraQueryBuilder` mapping each property of the OSLC resources to its counterpart in the Jira issue. Example of the mapping can be seen in Figure 6.1.

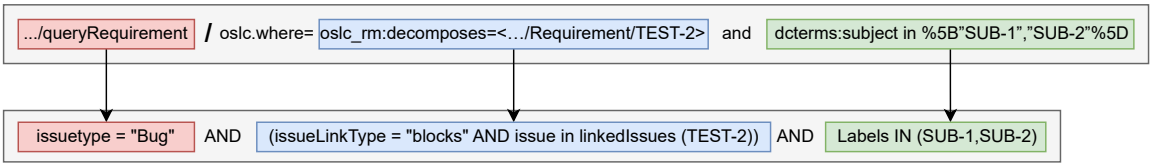

Figure 6.1: Visualisation of a simple `oslc.where` query mapping to JQL (`oslc.prefix` is omitted for brevity)

R4J Resources

The R4J API does not support any query language resulting in the inability to retrieve the data corresponding to the requested `jira_r4j:Folder` resources only. To overcome this limitation, a non-ideal but functional solution was implemented – retrieve the whole folder tree structure and filter the results on the adaptor side. This solution is not ideal as it requires the adaptor to retrieve a potentially enormous amount of data, which can result in performance issues.

To mitigate the performance issue, the adaptor could cache the retrieved folder tree structure and only retrieve it if the structure was modified since the last retrieval. The problematic part of this solution is how to determine if the structure was modified. The R4J API does not provide this metadata for the folder tree structure. A suitable solution might be to check if any issue from the project with the `modifed` timestamp greater than the timestamp of the cached tree structure exists, regardless if they belong to the tree structure. However, this would require investigation if the performance of this solution

would be better than the performance of the non-cached approach. Either way, the extent of this solution is beyond the scope of this thesis.

6.7 Bugs and Limitations

During the implementation, several bugs, issues, and limitations were discovered. This section goes over them and describes how they were resolved or mitigated.

R4J API Bug

An issue was discovered during the implementation of the folder update functionality. The R4J API returns `500 - Internal Server Error` when updating a folder with the `parentFolder` set to the `ROOT` of the directory structure, even though the folder is updated correctly. This issue was reported to the R4J API team, and a temporal workaround was implemented in the adaptor. The temporary solution ignores the `500 - Internal Server Error`, validates if the folder was updated correctly, and if validation passes, the update is considered successful.

Clients memory leak

During the testing of the adaptor, a memory leak was discovered in the clients. The leak was caused by the clients not closing the HTTP connections after the request was completed. It is unknown if this bug is in the implementation provided by the Jira API SDK or if it was caused by the extensions built on top of it. The created solution caches references to all the connections opened by the clients during the processing of a single request to the adaptor and closes them after the request is completed.

Contributors

Both the definitions `Requirement` and `RequirementCollection` contain the `contributors` property. The Jira API exposes a `changelog` property in the issue resource, which contains the information about changes done to the issue and who performed them. However, there is a limitation in the Jira API, which does not allow the retrieved changelog to exceed 500 entries. This can result in the `contributors` property not containing all the contributors for issues with a large number of changes. Currently, there is no solution for this issue.

Issue types

Some `issueTypes` require additional fields to be set when creating the issue (e.g., `Epic` requires the `Epic Name` field to be set). The adaptor has no way to verify if the custom required fields are set. Thus, the create operation can fail and result in a `500 - Internal Server Error`.

A problem arises when the `issueTypes` specified in the configuration file for both `Requirement` and `RequirementCollection` are the same. This results in the adaptor not being able to distinguish between the two resource types and thus returning the `Requirement` resources as `RequirementCollection` resources and vice versa.

It is left up to the users of the adaptor to ensure a valid `issueTypes` are specified in the configuration file, which does not require any additional fields to be set.

Chapter 7

Evaluation and Testing

This chapter contains the evaluation of the implemented adaptors and used technologies, as well as an overview of the testing of both adaptors.

7.1 Adaptors Testing

The created adaptors fully comply with the OSLC Core and Requirement Management specifications by providing all of the required resources, properties, and functionality. The Jira adaptor can be used alone to fully manage requirements in Jira. However, there is no way to differentiate between requirements and other issues as long as they have the same `issueType`. The R4J adaptor provides a solution for this by allowing the user to retrieve all of the requirements based on if they were added to the R4J folder tree structure.

The Jira and R4J adaptors have been end-to-end tested using the *Postman* [50] tool. The testing was done by creating a collection of requests, which tests the basic and query functionality of the adaptors. The collection is comprised of 361 HTTP requests and 446 tests. The collection can be exported as a JSON file and used by a continuous integration tool to validate that the changes to adaptors did not introduce any functionality breaks. An example of the flow of the test validating the problematic update functionality for folders with `parentFolder` set to `ROOT` discussed in Section 6.7 can be seen in Figure 7.1. The full collection of tests can be found in the `test/postman` directory in the source code.

To further validate the functionality of the Jira adaptor, a Python client for the adaptor has been created. This client provides a simple interface for importing and exporting requirements from Jira in the ReqIF format through the adaptor by utilizing the `rdflib` [55] and `reqif` [56] python libraries. The importing and exporting have been successfully tested by an example ReqIF file provided by Honeywell, testing the functionality of the adaptor in a real-world use case.

7.2 OSLC Evaluation

The OSLC Core specification is well-written and easy to follow. However, the OSLC Requirement Management specification is not detailed enough, which significantly impacts its comprehensibility. The specification does not provide any examples of how the resources should look nor any diagrams explaining the relations between the resources. Compared to the OSLC Automation Management specification, which provides both examples and diagrams, the OSLC Requirement Management specification could use some improvements.

The Eclipse Lyo tooling greatly reduces the complexity and time cost of creating an OSLC adaptor. However, the initial time investment in setting up the tooling is quite high, as well as the learning curve of the tooling. The documentation for the tooling is outdated and does not reflect the current state of the tooling. As a result, it is really hard to find any information about the tooling and how to use it. The learning curve of the tooling could be greatly reduced by providing a video or text-based tutorial detailing the process of creating an adaptor from start to finish.

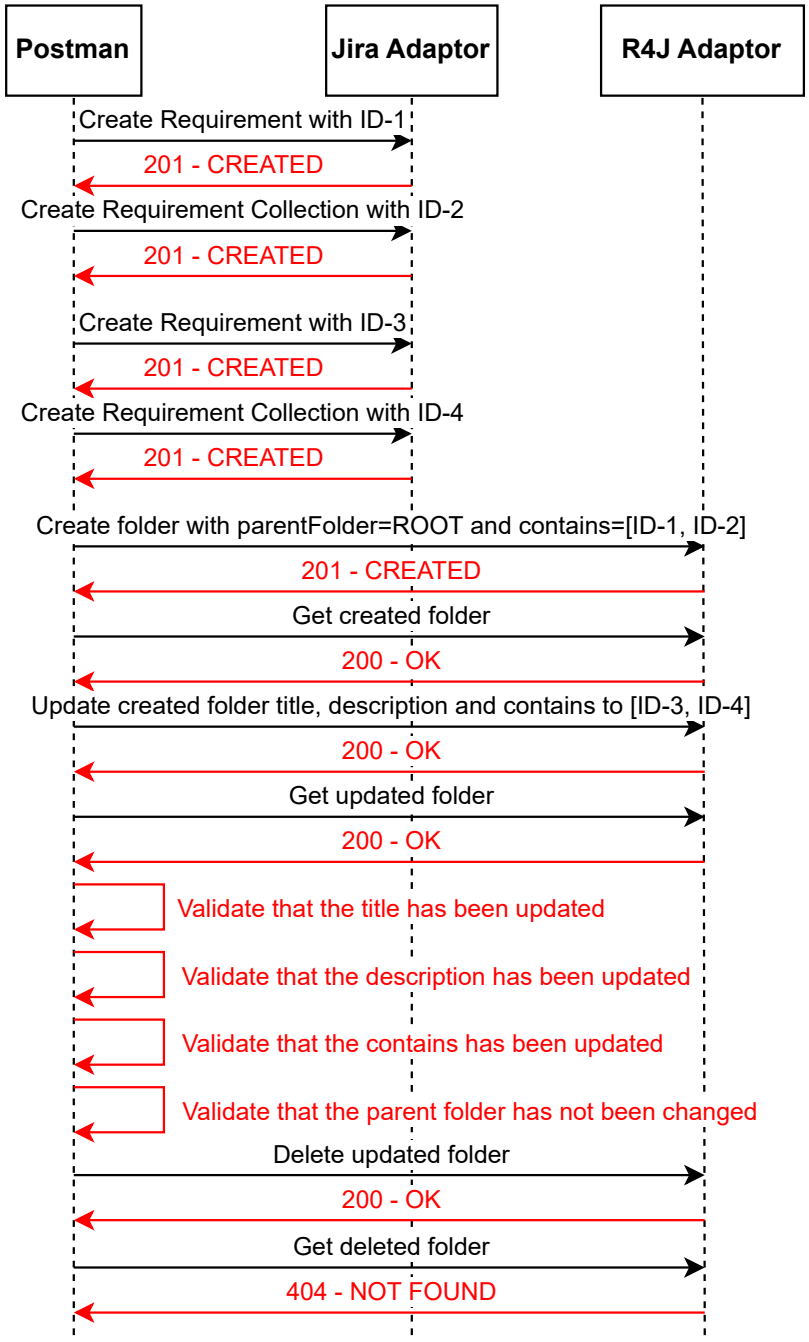

Figure 7.1: Example of the test flow for the Folder Update (*Red – test assertions*)

7.3 Jira and R4J Evaluation

The creation of the OSLC adaptors for Jira and R4J has been greatly impacted by the quality of the Jira and R4J API. The Jira API is well-designed and well-documented, providing a solid foundation for the creation of the Jira adaptor. The R4J API, on the other hand, lacks documentation, is designed a bit insufficiently, and is missing some important features, such as a query capability on the folder endpoint.

The development has also been greatly accelerated by the use of the Jira API SDK, which provides a great base for the creation of clients for the Jira API.

Chapter 8

Conclusion

The goal of this thesis was to provide unified access to requirements stored in Jira and R4J through the OSLC interface. For this purpose, the OSLC Core and Requirement Management specifications were first described in the context of Jira and R4J, and an overview of the tools that can be utilized to aid with the OSLC adoption was given. Afterward, a detailed description of the whole adaptor creation process was provided.

Two adaptors have been designed and implemented as a part of this thesis – the additional Jira adaptor, which fully satisfies the OSLC Core and Requirement Management specifications, and the R4J adaptor, which provides the folder functionality from R4J. The adaptors have been tested and evaluated by a collection of 361 HTTP requests using the Postman test suit, proving their functionality and usability. Moreover, a Python command line client has been created for the Jira adaptor, providing a simple interface for importing and exporting requirements from Jira in the ReqIF format.

The generic implementation of the adaptors implementation allows for the work to be continued and built upon in the future. Some of the possible improvements for future work include:

- **Folder query optimalization** – the optimalization of the `folder` query by using caching as discussed in Section 6.6
- **Generic Requirement Management adaptor** – the creation of a generic adaptor, which would be able to work with any Requirement Management tool
- **Full OSLC Query support** – the addition of the full OSLC Query support to the adaptors by creating a parser for the full extent of the OSLC Query language

Bibliography

[1] *Apache Maven* [online]. [cit. 2023-04-26]. Available at: https://maven.apache.org/.

[2] *Atlassian* [online]. [cit. 2023-04-07]. Available at: https://www.atlassian.com/.

[3] *Dive Into Design Patterns – Facade Design Pattern* [online]. [cit. 2023-04-29]. Available at: https://refactoring.guru/design-patterns/facade.

[4] *Dive Into Design Patterns – Singleton Design Pattern* [online]. [cit. 2023-04-29]. Available at: https://refactoring.guru/design-patterns/singleton.

[5] *Dive Into Design Patterns – Strategy Design Pattern* [online]. [cit. 2023-04-26]. Available at: https://refactoring.guru/design-patterns/strategy.

[6] *Dublin Core* [online]. [cit. 2023-04-27]. Available at: http://purl.org/dc/terms/.

[7] *Eclipse* [online]. [cit. 2023-04-07]. Available at: https://www.eclipse.org/.

[8] *Eclipse Jetty* [online]. [cit. 2023-04-26]. Available at: https://www.eclipse.org/jetty/.

[9] *Eclipse Lyo* [online]. [cit. 2023-04-07]. Available at: https://www.eclipse.org/lyo/.

[10] *Eclipse Lyo – Toolchain Designer & Lyo Store Tutorial* [online]. [cit. 2023-04-26]. Available at: https://www.youtube.com/watch?v=tZxPzlSTdeM.

[11] *Experimental OSLC Bugzilla application* [online]. [cit. 2023-04-26]. Available at: https://github.com/OSLC/lyo-adaptor-bugzilla.

[12] Fielding, R. T. *Architectural Styles and the Design of Network-based Software Architectures*. Irvine, CA, 2000. Dissertation. University of California, Irvine. Available at: https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm.

[13] *Friend of a Friend* [online]. [cit. 2023-04-27]. Available at: http://xmlns.com/foaf/0.1/.

[14] Hood, C., Wiedemann, S., Fichtinger, S. and Pautz, U. *Requirements Management: The Interface Between Requirements Development and All Other Systems Engineering Processes*. 2008th ed. Springer, 2007. ISBN 978-3540476894.

[15] *Hypertext Transfer Protocol – HTTP/1.0* [online]. [cit. 2023-04-22]. Available at: https://www.w3.org/Protocols/HTTP/1.0/spec.html.

[16] *IBM* [online]. [cit. 2023-04-21]. Available at: https://www.ibm.com/us-en/.

[17] *IBM Engineering Requirements Management DOORS Next* [online]. [cit. 2023-04-06]. Available at: https://www.ibm.com/products/requirements-management-doors-next.

[18] IEEE. ISO/IEC/IEEE International Standard – Systems and software engineering Vocabulary. 2017, [cit. 2023-04-20]. Available at: https://standards.ieee.org/ieee/24765/6800/.

[19] *Jira* [online]. [cit. 2023-04-07]. Available at: https://www.atlassian.com/software/jira.

[20] *Jira API Java SDK* [online]. [cit. 2023-04-29]. Available at: https://developer.atlassian.com/server/jira/platform/java-apis/.

[21] *Jira Query Language* [online]. [cit. 2023-04-30]. Available at: https://support.atlassian.com/jira-service-management-cloud/docs/use-advanced-search-with-jira-query-language-jql/.

[22] *Jira REST API* [online]. [cit. 2023-04-20]. Available at: https://developer.atlassian.com/cloud/jira/software/rest/intro/.

[23] *Jira Software Server Download* [online]. [cit. 2023-05-04]. Available at: https://www.atlassian.com/software/jira/update.

[24] *Linked Data* [online]. [cit. 2023-04-06]. Available at: https://www.w3.org/standards/semanticweb/data.

[25] *Linked Data Design Issues* [online]. [cit. 2023-04-06]. Available at: https://www.w3.org/DesignIssues/LinkedData.html.

[26] *Lyo Designer* [online]. [cit. 2023-04-07]. Available at: https://oslc.github.io/developing-oslc-applications/eclipse_lyo/lyo-designer.html.

[27] *Maven OSLC4J* [online]. [cit. 2023-04-07]. Available at: https://mvnrepository.com/artifact/org.eclipse.lyo.oslc4j.core/oslc4j-core/4.1.0.

[28] *OASIS Open Project* [online]. [cit. 2023-04-06]. Available at: https://www.oasis-open.org/.

[29] *OAuth 1.0 Revision A* [online]. [cit. 2023-05-08]. Available at: https://oauth.net/core/1.0a/.

[30] *Open Services for Lifecycle Collaboration* [online]. [cit. 2023-04-05]. Available at: https://open-services.net/.

[31] *OSLC Connector for Jira* [online]. [cit. 2023-04-21]. Available at: https://marketplace.atlassian.com/apps/1222299/oslc-connector-for-jira.

[32] *OSLC Core Delegated UI* [online]. [cit. 2023-04-06]. Available at: https://docs.oasis-open-projects.org/oslc-op/core/v3.0/os/dialogs.html.

[33] *OSLC Core Discovery* [online]. [cit. 2023-04-06]. Available at: https://docs.oasis-open-projects.org/oslc-op/core/v3.0/os/discovery.html.

[34] *OSLC Core Error* [online]. [cit. 2023-04-07]. Available at: https://docs.oasis-open-projects.org/oslc-op/core/v3.0/os/core-vocab.html#Error.

[35] *OSLC Core Query* [online]. [cit. 2023-04-06]. Available at: https://docs.oasis-open-projects.org/oslc-op/query/v3.0/os/oslc-query.html.

[36] *OSLC Core Resource Shape* [online]. [cit. 2023-04-06]. Available at: https://docs.oasis-open-projects.org/oslc-op/core/v3.0/os/resource-shape.html.

[37] *OSLC Core Specification 3.0* [online]. [cit. 2023-04-06]. Available at: https://docs.oasis-open-projects.org/oslc-op/core/v3.0/os/oslc-core.html.

[38] *OSLC Domain Modelling Workshop* [online]. [cit. 2023-04-26]. Available at: https://oslc.github.io/developing-oslc-applications/eclipse_lyo/domain-specification-modelling-workshop.html.

[39] *OSLC Lyo Adaptor Sample Modelling* [online]. [cit. 2023-04-26]. Available at: https://github.com/OSLC/lyo-adaptor-sample-modelling.

[40] *OSLC Lyo Domains* [online]. [cit. 2023-04-26]. Available at: https://github.com/eclipse/lyo/tree/master/domains.

[41] *OSLC Primary Integration Techniques* [online]. [cit. 2023-04-06]. Available at: https://open-services.net/resources/oslc-primer/#primary-oslc-integration-techniques.

[42] *OSLC Primer* [online]. [cit. 2023-04-06]. Available at: https://open-services.net/resources/oslc-primer.

[43] *OSLC Requirement Management Requirement* [online]. [cit. 2023-04-07]. Available at: http://open-services.net/ns/rm#Requirement.

[44] *OSLC Requirement Management Requirement Collection* [online]. [cit. 2023-04-07]. Available at: http://open-services.net/ns/rm#RequirementCollection.

[45] *OSLC Requirement Management Requirement Collection Constraints* [online]. [cit. 2023-04-07]. Available at: https://docs.oasis-open-projects.org/oslc-op/rm/v2.1/os/requirements-management-shapes.html#h3_RequirementCollectionShape.

[46] *OSLC Requirement Management Requirement Constraints* [online]. [cit. 2023-04-07]. Available at: https://docs.oasis-open-projects.org/oslc-op/rm/v2.1/os/requirements-management-shapes.html#h3_RequirementShape.

[47] *OSLC Requirements Management Specification 2.1* [online]. [cit. 2023-04-06]. Available at: https://docs.oasis-open-projects.org/oslc-op/rm/v2.1/os/requirements-management-spec.html.

[48] *OSLC Toolchain Modelling Workshop* [online]. [cit. 2023-04-26]. Available at: https://oslc.github.io/developing-oslc-applications/eclipse_lyo/toolchain-modelling-workshop.html.

[49] *OSLC4Net Github* [online]. [cit. 2023-04-26]. Available at: https://github.com/OSLC/oslc4net.

[50] *Postman* [online]. Available at: https://www.postman.com/.

[51] *Proof Key for Code Exchange by OAuth Public Clients* [online]. [cit. 2023-04-22]. Available at: https://datatracker.ietf.org/doc/html/rfc7636.

[52] *R4J – Requirements for Jira* [online]. [cit. 2023-04-20]. Available at: https://marketplace.atlassian.com/apps/1213064/r4j-requirements-management-for-jira.

[53] *R4J – Requirements for Jira REST API* [online]. [cit. 2023-04-20]. Available at: https://easesolutions.atlassian.net/wiki/spaces/REQ4J/pages/1490616345/REST+API+2.0.

[54] *RDF Primer* [online]. [cit. 2023-04-22]. Available at: https://www.w3.org/TR/rdf11-concepts/.

[55] *RDFLib* [online]. [cit. 2023-05-01]. Available at: https://rdflib.readthedocs.io/en/stable/.

[56] *ReqIF* [online]. [cit. 2023-05-01]. Available at: https://pypi.org/project/reqif/.

[57] *Requirements Interchange Format (ReqIF)* [online]. [cit. 2023-04-21]. Available at: https://www.omg.org/spec/ReqIF/1.2/PDF.

[58] *Resource Description Framework* [online]. [cit. 2023-04-05]. Available at: https://www.w3.org/RDF/.

[59] *Resource Description Framework* [online]. [cit. 2023-04-27]. Available at: http://www.w3.org/1999/02/22-rdf-syntax-ns#.

[60] *Resource Description Framework Schema* [online]. [cit. 2023-04-27]. Available at: http://www.w3.org/2000/01/rdf-schema#.

[61] *Running Jira applications over SSL or HTTPS* [online]. [cit. 2023-05-04]. Available at: https://confluence.atlassian.com/adminjiraserver/running-jira-applications-over-ssl-or-https-938847764.html.

[62] *The Base16, Base32, and Base64 Data Encodings* [online]. [cit. 2023-04-22]. Available at: https://www.rfc-editor.org/rfc/rfc4648.html.

[63] *The 'Basic' HTTP Authentication Scheme* [online]. [cit. 2023-04-21]. Available at: https://datatracker.ietf.org/doc/html/rfc7617.

[64] *The OAuth 1.0 Protocol* [online]. [cit. 2023-04-22]. Available at: https://www.rfc-editor.org/rfc/rfc5849.

[65] *The OAuth 2.0 Authorization Framework* [online]. [cit. 2023-04-21]. Available at: https://www.rfc-editor.org/rfc/rfc6749.

[66] *Turtle* [online]. [cit. 2023-04-06]. Available at: https://www.w3.org/TR/turtle/.

[67] *Uniform Resource Identifier (URI): Generic Syntax* [online]. [cit. 2023-04-06]. Available at: https://www.rfc-editor.org/rfc/rfc2396.

[68] *VeriFIT – Unite* [online]. [cit. 2023-04-26]. Available at: https://pajda.fit.vutbr.cz/verifit/unite.

[69] *World Wide Web Consortium* [online]. [cit. 2023-04-06]. Available at: https://www.w3.org/.

Appendix A

Setup and Usage manual

This chapter covers the installation and usage of the adaptors for Jira and R4J on the Linux operating system.

A.1 Jira Installation

The Jira Software Server can be donwloaded from the Atlassian website [23] as a tar.gz archive. The installation is done by extracting the archive to a directory of choice – this directory will be referred to as `JIRA_INSTALL_DIR`. Before running Jira, it is necessary to configure the folder, in which the data will be stored. This is done by environment variables `JIRA_HOME` – the configured forlder will be referred to as `JIRA_HOME_DIR`. The Jira server can now be started by running the `JIRA_INSTALL_DIR/bin/start-jira.sh` script and will be available on `localhost:8080` after the startup is completed. The Jira server can be stopped by running the `JIRA_INSTALL_DIR/bin/stop-jira.sh` script.

A.2 R4J Installation

The Requirements for Jira plugin can be added to the Jira instance by any administrator account by installing it from the Atlassian Marketplace found in Administration → Manage apps.

A.3 SSL Configuration

The BASIC authentication method does not require communication over HTTPS and be therefore used without any additional configuration. However, one of the requirements presented by the OAuth2 standard is for the communication to be done over HTTPS, so if this method is to be used, it is necessary to configure the Jira server to use TLS.

Jira SSL Configuration

The Jira server can be configured to use TLS by following the official Atlassian documentation [61]. The configuration is done by creating a `KeyStore` in the `JIRA_HOME_DIR` directory, generating a certificate and configuring the Jira server to use specified `KeyStore` in `JIRA_INSTALL_DIR/conf/server.xml` file. After the configuration is done, the Jira server has to be restarted. In order for the OAuth redirect to work, it is necessary to change the

`Base URL` in the Jira Administration → System → General Configuration to use HTTPS instead of HTTP.

Adaptors SSL Configuration

The adaptors now have to be configured to use the same certificate as creted in the previous step. The easiest way to do this is to download the certificate from the browser and import it into the Java `KeyStore` used by the adaptors. The `KeyStore` is located in `JAVA_HOME/lib/security/cacerts` with the default password `changeit`. The certificate can be added by running the following command:
`keytool -import -noprompt -alias localhost -file  /Downloads/localhost.pem -keystore default-java/lib/security/cacerts -storepass changeit`

A.4 Adaptors Configuration

The adaptors have to be configured before they can be used. The configuration is done by providing a JSON file in the `config` folder with the name `configuration.json`. The structure and meaining of each of the configuration options is described in a `config/README.md` file.

A.5 Adaptors Usage

Build and execution scripts are provided with the adaptors. The adaptors can be started by running the `run_jira.sh` to start only the Jira adaptor, or `run_all.sh` to start both adaptors. Both of these scripts support the `-b` flag, which builds the adaptors before running them. Upon startup the presence of the configuration file is checked, the adaptors are started and validated for successful startup. The adaptors can be stopped by pressing `Ctrl+C` in the terminal window, in which they are running. By default he Jira adaptor is started on `localhost:8081` and the R4J adaptor on `localhost:8082`, but these ports can be changed in the `poml.xml` files of the adaptors.

Appendix B

Contents of the included storage media

- `code` – source code of the adaptors, including the build and execution scripts, configuration files, README.md files and python ReqIF client
- `doc` – thesis text in PDF format and source LaTeXfiles
- `libs` – libraries used by the adaptors and the ReqIF client