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

Abstract

Do tohoto odstavce bude zapsán výtah (abstrakt) práce v anglickém jazyce.

Abstrakt

Do tohoto odstavce bude zapsán výtah (abstrakt) práce v českém (slovenském) jazyce.

Keywords

Sem budou zapsána jednotlivá klíčová slova v anglickém jazyce, oddělená čárkami.

Klíčová slova

Sem budou zapsána jednotlivá klíčová slova v českém (slovenském) jazyce, oddělená čárkami.

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

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana X... Další informace mi poskytli... Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Daniel Pindur
April 23, 2023

Acknowledgements

V této sekci je možno uvést poděkování vedoucímu práce a těm, kteří poskytli odbornou pomoc (externí zadavatel, konzultant apod.).

Contents

Chapter 1

Introduction

This chapter summarizes the motivation and objectives of this thesis, the solution taken to achieve the objectives, and the results of the thesis. In the last section, the structure of the thesis is presented.

1.1 Motivation and Objectives

This work was initiated by Honeywell International Inc. [5], which currently uses IBM Doors as a solution for requirement management. The company is exploring the option and considering switching to a new solution – Jira R4J. Honeywell currently provides its clients access to requirements stored in IBM Doors via the OSLC interface for Requirement Management Specification, which comes with IBM Doors. The company is interested in providing the same access to requirements stored in Jira R4J. The goal of this thesis is to explore and create a solution that will allow Honeywell to provide its clients with 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

[[Write after thesis structure is finalized]]

Chapter 2

Background

This chapter briefly covers the basics of requirements management and its fundamental concepts, as well as 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 management 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 [6].

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 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 [10].

They can be divided into two main categories, *functional* and *non-functional* 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).

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 [9] is a requirement management system developed by IBM [8]. 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. All of the previously mentioned features are available through the web user interface, REST API as well as native OSLC interface.

Jira Software and Requirements for Jira

Jira Software [11] is an issue-tracking and project management tool developed by Atlassian [1]. 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 [12] with BASIC or OAuth2 authentication, further described in chapter 3. 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 [20] for Change Management Specification).

Requirements for Jira (R4J) [35] 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 [36]. 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) [38] 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

Chapter 3

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.

3.1 Basic Access Authentication

BASIC (Basic Access Authentication) was first defined as a part of HTTP 1.0 specification [7], but the standard has been since superseded and redefined as part of its own RFC [41]. 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 [40], 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.

3.2 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 authentication 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 [42], which used asymmetric cryptography to encrypt and verify the credentials, but was later superseded by OAuth 2.0 [43], 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.

3.2.1 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.

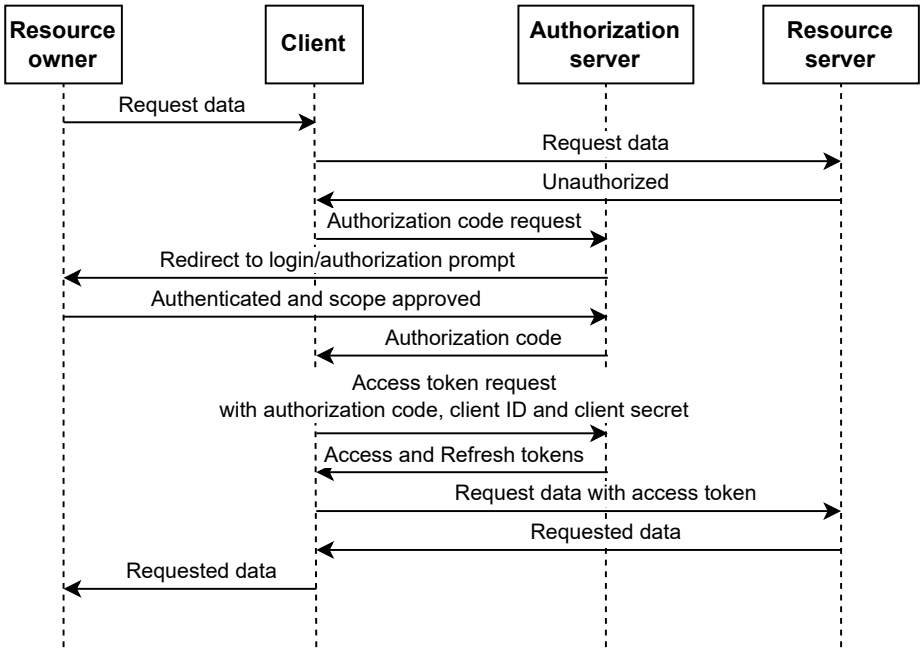

Figure 3.1: OAuth 2.0 Authorization Code Grant flow diagram

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.

Proof Key for Code Exchange

PKCE (Proof Key for Code Exchange) [34] 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. 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.

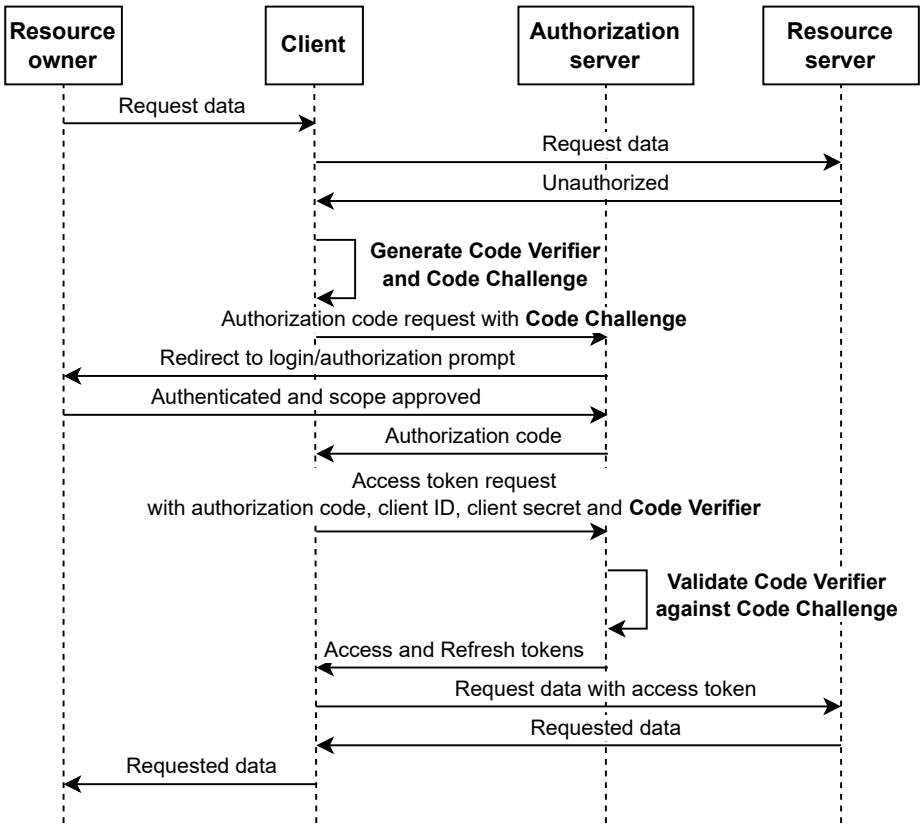

Figure 3.2: OAuth 2.0 Proof Key for Code Exchange flow diagram

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.

OpenID Connect

OIDC (OpenID Connect) [19] is an extension of OAuth 2.0, which allows the authentication of a user against the authorization server as well as obtaining basic user profile information about the authenticated user.

Chapter 4

OSLC – Open Services for Lifecycle Collaboration

This chapter provides a brief overview of the OSLC (Open Services for Lifecycle Collaboration) [18], its fundamental technologies, and the Core and Requirement Management specifications, which are used in this thesis.

4.1 Overview

OSLC [18] is an OASIS Open Project [17] 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* [28].

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* [27].

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 4.2. 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 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.

4.2 Fundamental Technologies

OSLC is built on top of several fundamental technologies. This section lists these technologies and introduces briefly each of them.

REST

REST (REpresentational State Transfer) [4] 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).

RDF

RDF (Resource Description Framework) [39] is a W3C (World Wide Web Consortium) [46] 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. 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 [44], RDF/XML, and RDF/JSON.

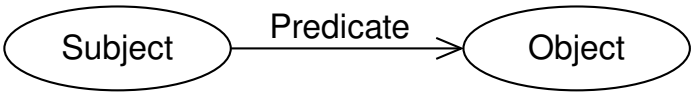

Figure 4.1: RDF Graph Triplet (source: [37])

Linked Data

Linked Data [13] 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 [14]:

1. URIs [45] are used as names to identify things
2. People can lookup things using HTTTP 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

4.3 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.

4.3.1 OSLC Core Specification

At the time of writing this thesis, the current version of OSLC Core Specification [26] 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.

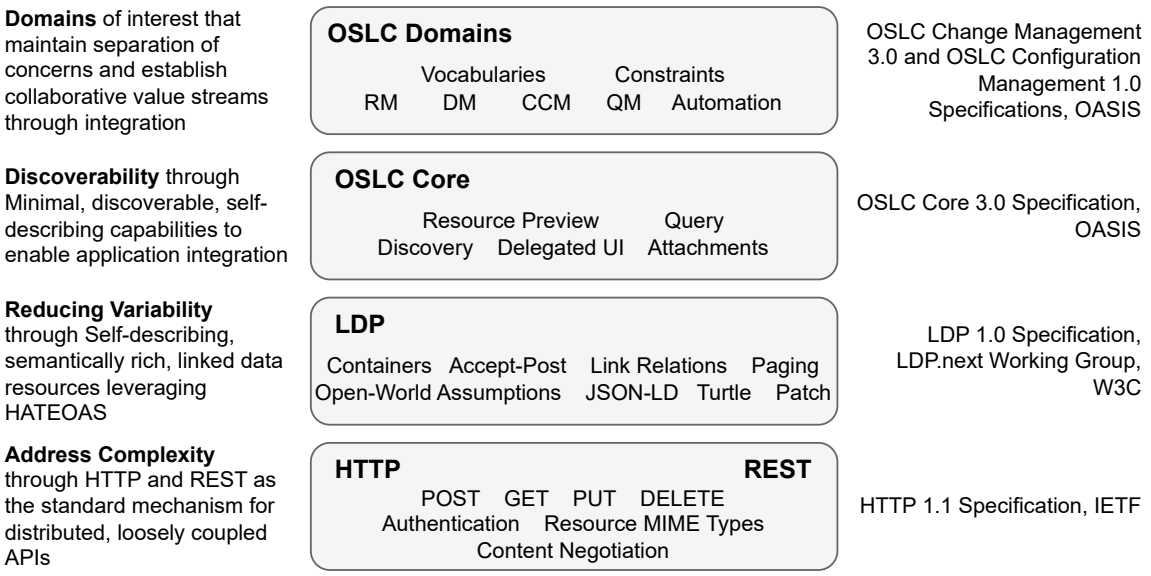

Figure 4.2: OSLC Core 3.0 architecture (source: [26], remade)

Resource Shape

OSLC works with resources, which are uniquely identified by a URI [45], and are represented by RDF triples. Resource Shape [25] is a resource of type `oslc:ResourceShape`, that describes the contents and constraints of other resources.

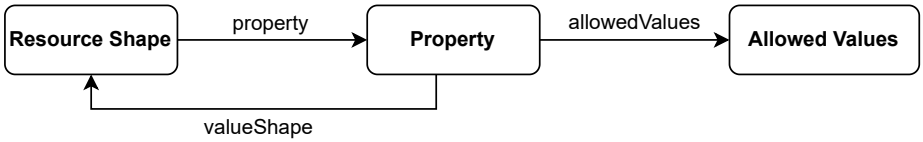

Figure 4.3: Resource Shape (source: [25], remade)

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.

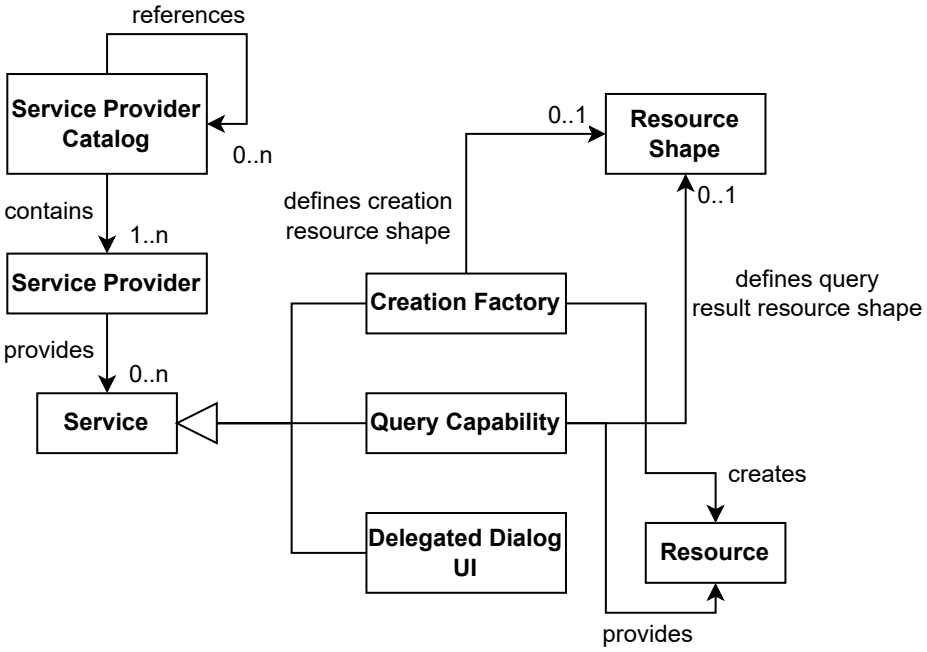

Figure 4.4: OSLC Discovery diagram (source: [22], remade)

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*. For additional information about the Discovery mechanism can be found in the Discovery section of the OSLC Core Specification [22].

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 [21] 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 4.1. 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 [24]. 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 4.1).

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. All error responses should be returned as a resource of type `oslc:Error` [23], which contains a human-readable message and a machine-readable error code. This enables clients to handle errors in a generic way.

4.3.2 OSLC Requirement Management Specification

At the time of writing this thesis, the current version of OSLC Requirement Management Specification [33] is 2.1. The specification builds on top of the OSLC Core Specification 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.

```
<?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 4.1: OSLC Query Result example

Requirement

`oslc_rm:Requirement` [29] is a resource shape used for describing a single requirement described in 2.1. Requirement Management Specification defines an extensive set of properties and constraints [32] 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` [30] 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 [31], but they are nearly the same as for `oslc_rm:Requirement` resource shape.

4.4 Eclipse Lyo

Eclipse Lyo [3] is an open-source project hosted by the Eclipse Foundation [2] 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.

OSLC4J SDK

OSLC4J Software Development Kit (available at Maven Repository [16]) 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 of resources to Java objects.

Lyo Designer

Lyo Designer [15] 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. 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 5

Adaptor Design

Chapter 6

Implementation

Chapter 7

Evaluation and Testing

Chapter 8

Conclusion

Bibliography

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

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

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

[4] 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.

[5] *Honeywell International Inc.* [online]. [cit. 2023-04-06]. Available at: https://www.honeywell.com/cz/en.

[6] 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.

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

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

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

[10] 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/.

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

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

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

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

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

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

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

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

[19] *OpenID Connect* [online]. [cit. 2023-04-22]. Available at: https://openid.net/connect/.

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

[21] *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.

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

[23] *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.

[24] *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.

[25] *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.

[26] *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.

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

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

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

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

[31] *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.

[32] *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.

[33] *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.

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

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

[36] *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.

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

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

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

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

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

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

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

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

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

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