Tampere University

Talha Manj

OSLC STANDARD IN THE VISUAL DIAGNOSIS FOR DEVOPS SOFTWARE DEVELOPMENT

Master Thesis
Faculty of Information Technology and Communication Sciences
Examiner: Prof. Kari Systä
June 2023

ABSTRACT

Talha Manj: OSLC standard in the Visual Diagnosis for DevOps Software Development
Master Thesis
Tampere University
Master's Degree Programme in Software, Web & Cloud
June 2023

This research investigates Open Services for Lifecycle Collaboration (OSLC) standard. The data was gathered by analysing OSLC specification documents. OSLC, an open standard based on Linked Data, RDF, HTTP, and REST provides solutions for tool integration and data interoperability. This study also explores the root technologies of OSLC. OSLC is motivated by the domain-driven scenario that covers different disciplines involved in the software development lifecycle such as requirement management, designing, issue management, version management, and testing. Moreover, the investigated data is presented based on the problem statement to describe possible solutions OSLC can offer.

This thesis studies the Visual Diagnosis for DevOps Software Development (VISDOM) project, a visualization platform. In software development, various teams are working with different disciplines and collaborate for quality deliverables. VISDOM aims to gather data from tools used in software development and allow data visualization. These tools are diverse in terms of architecture, data formats and are not meant to be interoperable. VISDOM provides stakeholders-specific dashboards, for example, managers, developers, and business analysts. Therefore the data from various data sources have to be served with multiple use cases. This complicates the data processing component of the VISDOM data management system named as data adaptor. Therefore, the data adaptor is encountering certain challenges such as the reusability of modules, linking of data on the semantic level, and addition of more data sources.

The findings indicate that OSLC can offer solutions for standardizing the data adaptor of VISDOM. This finding is further explained with the benefits and limitations to adopt OSLC. The findings are based on theoretical analysis and lack implementation.

Keywords: Data adaptor, OSLC, VISDOM-project, data linking

PREFACE

This thesis is mainly investigating OSLC standard, the idea was based on the challenges the VISDOM project is having.

I would like to thank my supervisor Prof. Kari Systä for providing me with the opportunity to work on this idea. I thank him for his valuable guidance and comments throughout the research process.

Tampere, 5th June 2023

Talha Manj

CONTENTS

ABBREVATIONS AND SYMBOLS

ALM	Application Lifecycle Management
API	Application Programming Interface
CM	Change Management
HTML	HyoerText Markup Language
HTTP	Hyper Text Transfer Protocol
ITIL	Information Technology Infrastructure Library
JSON	JavaScript Object Notation
LDP	Linked Data Platform
MIME	Multipurpose Internet Mail Extensions
OSLC	Open Services for Lifecycle Collaboration
RDF	Resource Descriptive Framework
REST	Representational State Transfer
SDLC	Software Development Lifecycle
SPARQL	SPARQL Protocol and RDF Query Language
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
VISDOM	Visual diagnosis for DevOps software development
W3C	World Wide Web Consortium
XML	Extensible Markup Language

1. INTRODUCTION

Today software development process is more organized and measurable with the help of frameworks named Software Development Lifecycle (SDLC). SDLC is a process that considers various stages of software development such as planning, designing, building, documenting, testing, deploying and maintaining, and provides guidelines for tasks to be performed at each stage [46]. Several SDLC methods and models have been used today in software development, for example, Agile, Waterfall, DevOps, Lean, Iterative, and Spiral. SDLC allows organizations to measure each stage and provide the opportunity to enhance the software development process.

Hence, the work in the software development lifecycle has been divided into specialized teams at each stage to ensure product quality and timely delivery. Therefore, data gathered from various teams can be presented for better decision-making and more innovation using data visualization technique to make the software development process more refined and less prone to failure. One such technique has been utilized by VISDOM (Visual diagnosis for DevOps software development). VISDOM gathers data from various sources involved in a software development process for analytical purposes. Thus, allows the managers or stakeholders to have a better overview of the data so that the actual cause can be addressed without any time wastage. In addition, through the use of VISDOM, the development process can be improved, and this will help with cost reduction and will save time, which provides the ability to spend time on innovation rather than finding the defects within the process. However, the tools used in the development process produce huge amounts of heterogeneous data that are not meant to be inter-operable, and as a result, problems such as relating data, data reusability, scalability, data efficiency, fast retrieval of data, and querying of data arise. Therefore, the data model of VISDOM is not standardized and is facing certain issues and limitations in executing its objective. Additionally, there are certainly other challenges, such as the tools used within the development lifecycle have been updating rapidly, and the data is not consistent in terms of architecture and format.

To overcome the above challenges, standardization is considered to be the finest solution for complex systems such as VISDOM. Standards are rules or ways agreed upon by a community in a particular area of expertise. Standards make the system addressable to all possible barriers and challenges within a particular field. Furthermore, standards can also act as a benchmark for a product like VISDOM. Standardization can play a key role in VISDOM adopting its objectives. Therefore, the data model of VISDOM needs to be standardised, for example, the accumulated data from various sources should have a common language to make data relatable using the same syntax, format, and vocabulary.

One such promising standard is Open Services for Life-cycle Collaboration (OSLC). OSLC is an open-source collaboration framework, utilized for tool integration and data collaboration [34]. Therefore, various companies have adopted this standard for the exchange of information to interconnect the teams involved in the software development process to ensure the speedy delivery of high-quality products. OSLC is a domain-driven standard based on the Semantic Web [7] concept and provides numerous solutions for collaboration goals. In SDLC, domains are grouped as various disciplines engaged in a software development process such as development, testing, operations, and administration. OSLC adopted different W3C (World Wide Web Consortium) standards such as REST (Representational State Transfer) [14], HTTP (Hyper Text Transfer Protocol) [15], and RDF (Resource Description Framework) [27] and collectively provide solutions to the requirements in a minimalistic way.

This thesis has three main aims regarding the VISDOM project. Firstly, to investigate the OSLC standard thoroughly, then consider the needs of the VISDOM project, and finally, summarize the possible solutions for implementation and applicability to the previously developed data model. The study also includes the advantages provided by the OSLC. Therefore, for this thesis, the following research questions have to be answered:

1. Why is OSLC a possible standard for VISDOM?
2. What are the applicability options for VISDOM to adopt OSLC?
3. What are the benefits and solutions VISDOM can avail by standardization?

VISDOM architecture has a separate data processing layer that fetches, processes and provides visualisation-ready data to the frontend side. The data processing is based on several components such as data fetchers responsible for fetching and saving raw data to a database. The other component named the data adaptor is getting data from the database and processing and linking data to be able to visualise. This makes the data adaptor the complex part of the VISDOM data management system and will be the focus of this study.

In recent years, some research and implementation have been done for data interoperability and tool integration in the software development lifecycle. This study investigates how previous studies have implemented the OSLC standard to perceive their success factors and challenges, and then summarizes the options for VISDOM to adopt the OSLC standard.

Chapter 2 is a theoretical overview of OSLC background concepts. This chapter gives a brief introduction to the technologies and standards upon which OSLC is based. These technologies are linked data, RDF, SPARQL, HTTP and REST. A piece of concise information about these technologies has been explained in this section.

Chapter 3 provides details on OSLC and its core capabilities and architecture. This chapter focuses on OSLC itself, its specifications, and its domains. In addition, all the concepts related to OSLC adaptors are described.

Chapter 4 explains the VISDOM data management system as the context of research and presents the research methods for this thesis. This chapter will describe the VISDOM project and its objectives, research questions, and research problems in detail. Furthermore, the research approach and data-gathering technique will be explained in this chapter.

Chapter 5 derives the findings from gathered data. Further, this section includes analysis based on the research problem and explain how OSLC can provide solution to each problem. Whereas, an example of OSLC-compliant VISDOM architecture is also proposed in this chapter.

Chapter 6 will conclude this study by presenting the concise view of this study.

2. OSLC BACKGROUND

2.1 Linked Data

Linked data is the main pillar of the semantic web also known as web 3.0, an idea proposed by Tim Berners-Lee, the founder of the Web. In 2001, Tim Berners-Lee defines the concept as "The Semantic Web is an extension of the current web, in which information is given well-defined meaning, better-enabling computers and people to work in cooperation"[7]. The semantic web building blocks were revealed in the Semantic Web Stack illustrated in figure 2.1, which gives an overview of the whole semantic web concept and its dependencies.

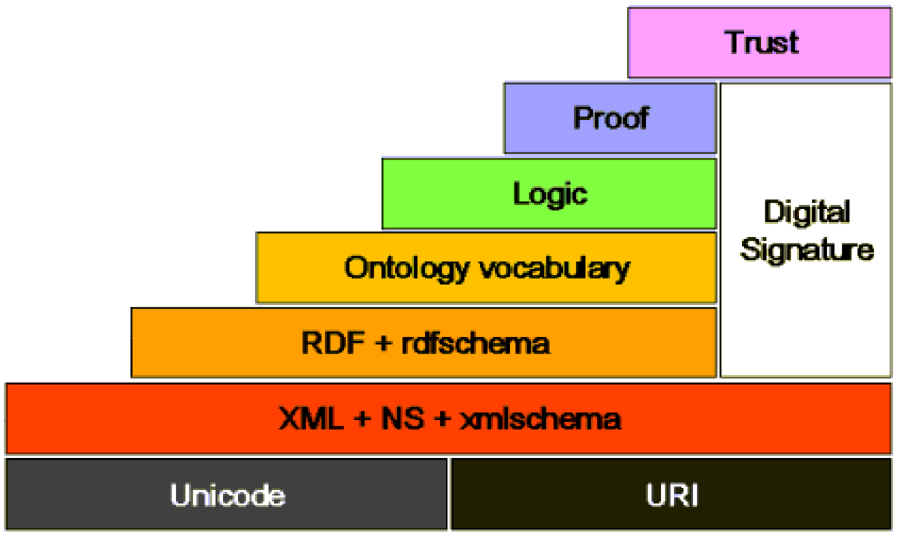


Figure 2.1. *Semantic Web Stack [49]*

Unlike Web 2.0 where data is in hypertext and is linked with a relationship anchor, linked data is about linking data using a Uniform Resource Identifier (URI) as labels and a Resource Description Framework (RDF) model as a representation of data. URI is a sequence of characters utilized to label or name a resource for interacting with other resources. A resource can be defined as a single entity or object, that needs a label to be accessible or to connect. Uniform Resource Locator (URL) that is used as a label to some web address is a common form of URI, but unlike URL, URI can or cannot be connected to a network [6]. Later in 2006, Tim Berners-Lee presented four rules by following them will make data linked data which are as follows [5]:

1. Use URIs as name of things
2. Use HTTP URIs so that people can look up those names
3. When someone looks up a URI, provide useful information, using the standards (RDF*, SPARQL)
4. Include links to other URIs so that they can discover more things

These points explain that any entity, object or resource within a data should be labelled with a URI. The second rule describes, that the data should have HTTP URIs and is available on the web. The labelling of resources with HTTP URIs allows data to be discovered or accessed uniquely without any duplication. The third rule says that data should be in triplet format following the RDF model and be connected to other data using classes and properties from ontologies such as RDFS or OWL. Therefore the relationship of triples can give proper semantics by following the used ontology. The fourth rule defines the URIs of data connected elsewhere to create a web of data, whether the data is not relevant to each other. Therefore, linked data properties applied on complex data sets make them more reachable and manageable in terms of relating data as it follows a particular RDF data model. Utilization of common data representation also increases the reusability of data, independent of its data format.

2.2 Resource Description Framework

One of the key elements of Semantic Web Stack is RDF (Resource Description Framework) as illustrated in Figure 2.1. This framework allows data to have relationships by the means of semantics. A resource could be anything of the user's interest for example it can be any object, individual, topic of interest, or some other entity [24]. RDF data model has a graph-like representation called RDF graph and is presented in triplets a subject, predicate and object. In RDF each resource has a set of triples, a subject, predicate and object, represented in figure 2.2 and an RDF graph consists of a collection of triplets.

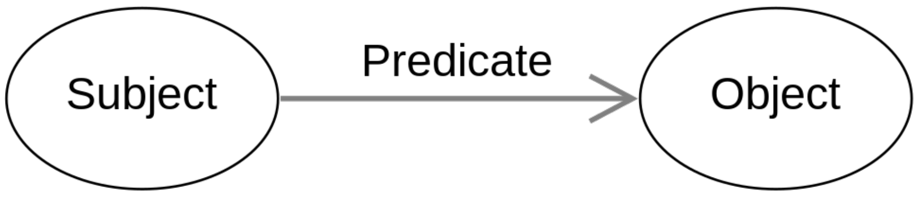


Figure 2.2. *RDF Triple [27]*

Whereas, in an RDF triple, a subject is any resource of interest and the predicate represents a relationship and an object could be another resource or literal (value of the particular resource). RDF is an open-source framework, therefore anyone can create properties of their own depending on the nature of their resource by following the rules of creating vocabularies [27]. In RDF triple, a subject has a URI reference, a predicate is labelled with a URI and the object is also a URI reference of some other resource or a literal (value of the subject). For example, in figure 2.3, for the birth date relation, as a person bob was born on some date which is the value of this relation and these are called literals, however for other relations such as Alice, it refers to another entity and is named as another object. As a result of this data merging and linking become easier [9]. RDF data model has an XML-based syntax and utilizes serialization of XML form in which a series of nodes have a sequence of elements inside them [20]. The language for writing RDF triples is called turtle language [3] which holds the textual representation of triples. The self-reliant property of RDF makes it easy to manipulate and process data for applications [27].

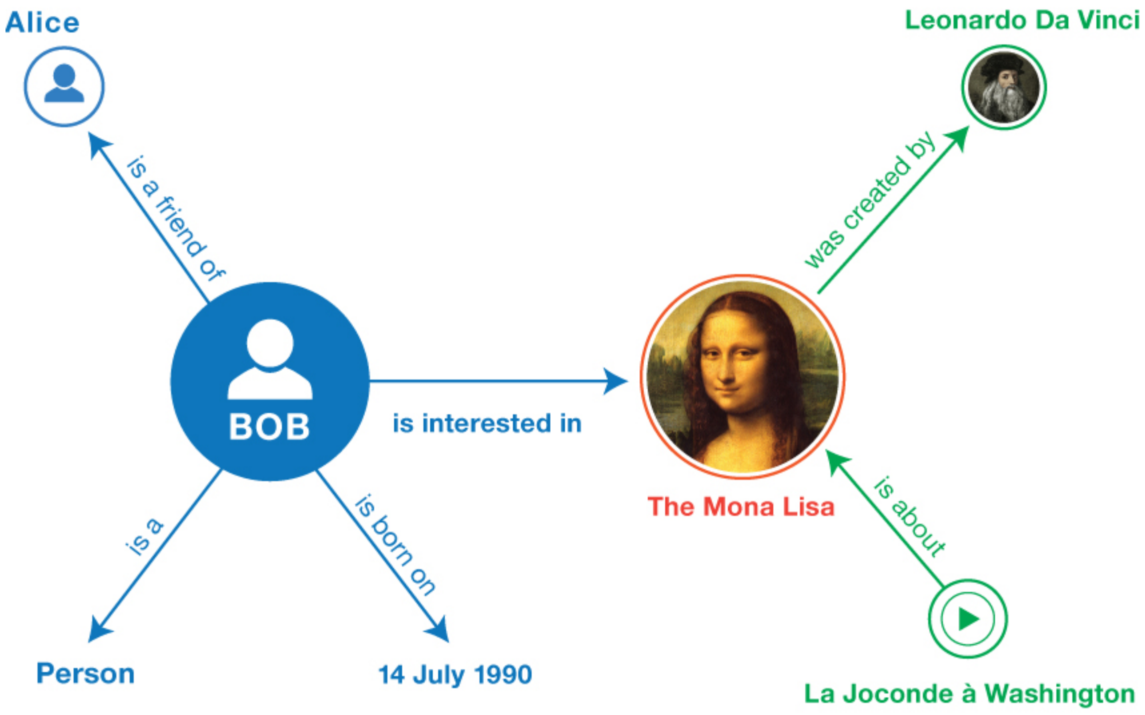


Figure 2.3. *RDF Graph example [45]*

The above Fig2.3 illustrates an RDF graph having a collection of RDF triples. In this example, bob is interested in The Mona Lisa resource which further extends to the resources related to The Mona Lisa and thus creates a collection of data through semantics. The below figure provides a URI representation of figure 2.3 where each entity has a URI and the relations between them are also represented with a URI. Thus having URI for each entity in an RDF triple allows data to be machine-readable and allows machines to interpret data by the means of semantics. Additionally, the URIs used for relations are defined in ontologies that define each entity of a property with proper definition and description.

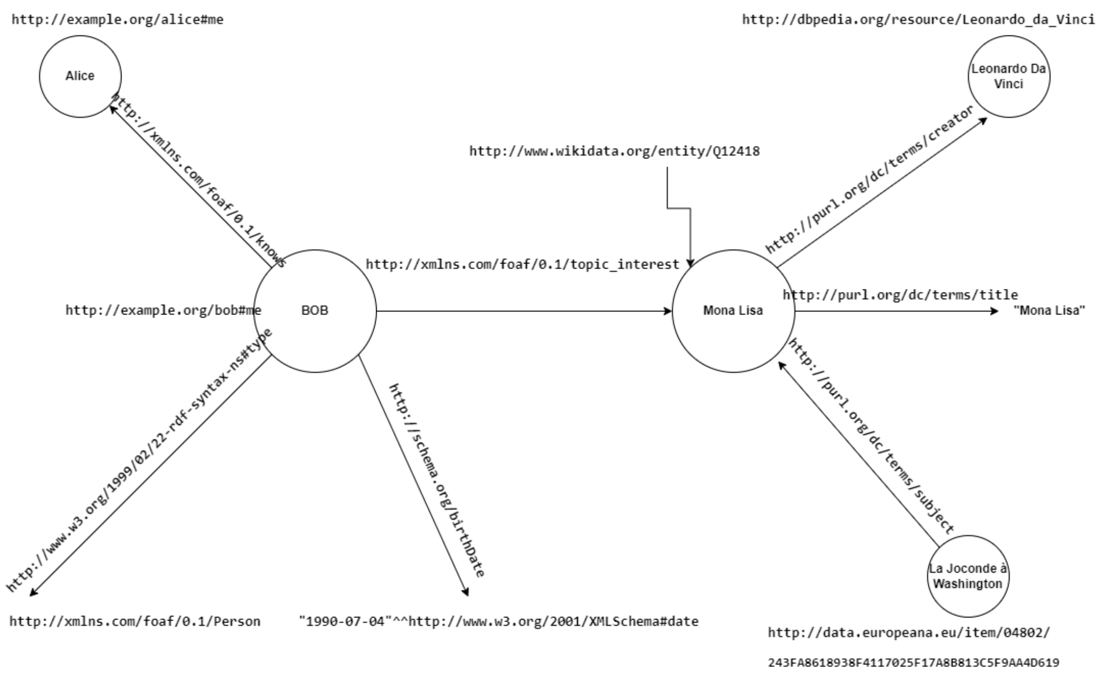

http://example.org/alice#me
Alice
http://xmlns.com/foaf/0.1/knows
http://example.org/bob#me
BOB
http://xmlns.com/foaf/0.1/topic_interest
http://www.wikidata.org/entity/Q12418
Mona Lisa
http://dbpedia.org/resource/Leonardo_da_Vinci
Leonardo Da Vinci
http://purl.org/dc/terms/creator
http://purl.org/dc/terms/title
"Mona Lisa"
http://purl.org/dc/terms/subject
La Joconde à Washington
http://data.europeana.eu/item/04802/
243FA8618938F4117025F17A8B813C5F9AA4D619
http://www.w3.org/1999/02/22-rdf-syntax-ns#type
http://xmlns.com/foaf/0.1/Person
http://schema.org/birthDate
"1990-07-04"^^http://www.w3.org/2001/XMLSchema#date

Figure 2.4. *URI representation of Figure 2.3*

One such vocabulary is RDF schema language called RDFs which describes the entities in classes, sub-classes and properties to make a relationship between resources. Many other vocabularies such as Web Ontology Language (OWL) [23], FOA[18], SKOS [47], Dublin Core[12], and schema.org[44]. Furthermore, RDF itself is a data model and can have various data format representations such as Turtle[3], JSON-LD[25], RDFa[45], RDF/XML[20]. Although the writing syntax would be different, however, these representation form triplets and make them logically equivalent [45].

2.3 SPARQL

SPARQL, the acronym for SPARQL Protocol and RDF Query Language, is a set of specifications designed by the W3C for RDF data models. The protocol part of SPARQL is a set of rules that describe how a client program and a SPARQL processing server exchange queries and how the result will be generated, which is a concern of SPARQL processor developers. The query language part of SPARQL gives a solution for querying data, whether the data is stored natively in RDF or just with RDF representation using some middle-ware [22]. SPARQL operates similarly to Structured Query Language (SQL) in a way, as it also deals with data having relations and allows relational algebra expressions to be executed [2]. RDF data has a complex structure as it is in the form of a graph linked with various resources such as information about things, personal information, social information, and the metadata of some resources. For that, SPARQL provides a solution for these disparate resources. SPARQL queries have three stages, the pattern matching stage is where the user can query data with several interesting features such as union, aggregation, nesting, filtering, execution of mathematical operations, and pattern matching. The second stage is the solution modifier, where the result of the first stage can be further optimized using operations such as ordering, limiting, distinction, and projecting to further advance the desired output. The last one is the output stage, in which the result from both the above executions is presented with a new RDF graph of values and descriptions that matches the query [43].

SPARQL searches for queries from data associated with human-readable information and presents the result in human-readable form. Although the query input takes a URI to execute, the processors are designed to look for human-readable information. SPARQL is very powerful due to a variety of options for a user to query data, such as using the OPTIONAL clause, putting filters for the desired condition, searching further in the data, eliminating duplication in the output using the DISTINCT keyword, combining conditions with Union, limiting the result, ordering the result, nested queries, and more [13].

2.4 Hypertext Transfer Protocol

The Hypertext transfer protocol (HTTP) is an application protocol that allows users to communicate with data on the web. HTTP is a request-response protocol that follows a client/server pattern. In addition to this HTTP generally uses Transmission Control Protocol (TCP)/ Internet Protocol (IP) for connection between a client and a server [15]. In HTTP architecture, an HTTP client sends a request through some web browser, search engine, command line, or any other application. Whereas, an HTTP server is an application that accepts the request and sends back the response to the client. HTTP request comprises of request-line, host URI, request header and request body [15]. The request line includes a method token that indicates the method to be performed on the resource such as (GET, POST, PUT and DELETE), a request-URI upon which the is to be sent along with the protocol version used for the connection. The host URI is the URI of the host by which the request has been initiated. Whereas the request header holds additional information about the request such as which media types are acceptable in response, for example, authorization credentials and further conditions for getting a response. An HTTP response is composed of a status line and response header [15], the status line includes a three-digit status code such as (100, 201, 404) and a reason phrase that is a short textual description of the status code. The response header field holds additional information about the response, it can have response content, media type details, authentication token, and necessary information about the server. In addition to these, a response could have a response body.

The HTTP protocol has been used widely for communication on the web due to its powerful features such as connection-less, media-independent, stateless, and extensible. HTTP/1.0 was connection-less, however, HTTP/1.1 the current version allows to have a persistent connection between the client and the server [15]. HTTP/1.1 provide an option to pipeline the connection, if the client wants to send multiple requests, as a result of this the computing time is reduced for both the client and the server due to having fewer connections. HTTP is media-independent, therefore any appropriate Multipurpose Internet Mail Extensions (MIME) type can be used for data communication between client and server, which makes HTTP extensible also, as the client and the server can agree on any data type to be exchanged.

MIME types are media types that indicate the nature and format of a file, document or stream of data. MIME types were now used widely to define the format of information exchange on the internet [19]. MIME-type mainly consists of two parts: a type and a sub-type separated by a slash ("/"). The first part type, represents the general category of the data upon which it falls such as text, video, and application. Whereas the sub-type further specifies the exact kind of data such as plain, XML, or HTML. For example, a type of text might have a sub-type plain, HTML, or calendar file. Similar to this example, there are various MIME types used to represent content on the internet such as text/plain, text/css, application/javascript, application/json etc.

HTTP allows caching which as a result eliminates the need to send a request again in some use cases and acts similarly in response situations [15]. Caching feature makes HTTP more efficient and helps to increase the application performance as well as improve user experience. Thus HTTP has been a widely used communication protocol due to its flexible, extensible, optimistic features.

2.5 REST

Representational State Transfer (REST) is an architectural pattern for network-based distributed systems and provides a set of constraints [14]. REST is a hybrid architectural style comprised of some previously used architectural styles such as Hierarchical Styles (Client-Server, Layered System) and adds some additional constraints to fulfil the requirements of modern Web architecture [17]. REST is not limited to web architecture, and can be applied to any other architecture by following the REST constraints. REST is not architecture itself, however, an architectural style that upon execution can provide certain benefits such as scalability, visibility, and flexibility [16]. The stateless behaviour of REST increases the scalability of the application, as the server does not have to save the state of the client requests and can handle several requests. The cache property of REST allows the requests to be cacheable on the client side and thus helps to increase the efficiency of the application by not sending the request again. This, as a result, decreases the latency of interaction in the application. REST uniform interface is the main constraint and permits the client from autonomous development. For example, to retrieve information about a user an endpoint GET API/user/id is called, to create a new user POST method is used in request and to update a resource PUT method is used. Although POST can also be used to update, the POST method creates a new entity while the PUT method act on an existing resource and is idempotent [15]. This indicates that upon each request proper representation should be used while communicating resources. Similar to the request, the response to a request should have a proper status code and representation of the requested resource. Furthermore, REST does not apply constraints on the resource itself. However, it describes a set of rules to access the resource.

3. OSLC CORE AND DOMAINS

3.1 Introduction to OSLC

Open Services for Life-cycle collaboration is an open standard which was initiated by IBM internally for its organization in 2008 with some other collaborators and the first minimal specifications were published in 2009. Later in 2013, the OSLC steering committee decided to make it open source and hand over its governance to OASIS [1].

OSLC is based on the world wide web consortium (W3C) standards and creates specifications allowing inter-operating data within the software development lifecycle. OSLC provides a solution for the needs of tool interoperability such as tools from software and hardware to link with a standardised approach. OSLC is motivated by domain-driven scenarios, and it allows the integration of heterogeneous data. It provides an architecture for interoperability that is standardized, minimalistic and loosely coupled. Figure 3.1 describe the architecture of OSLC.

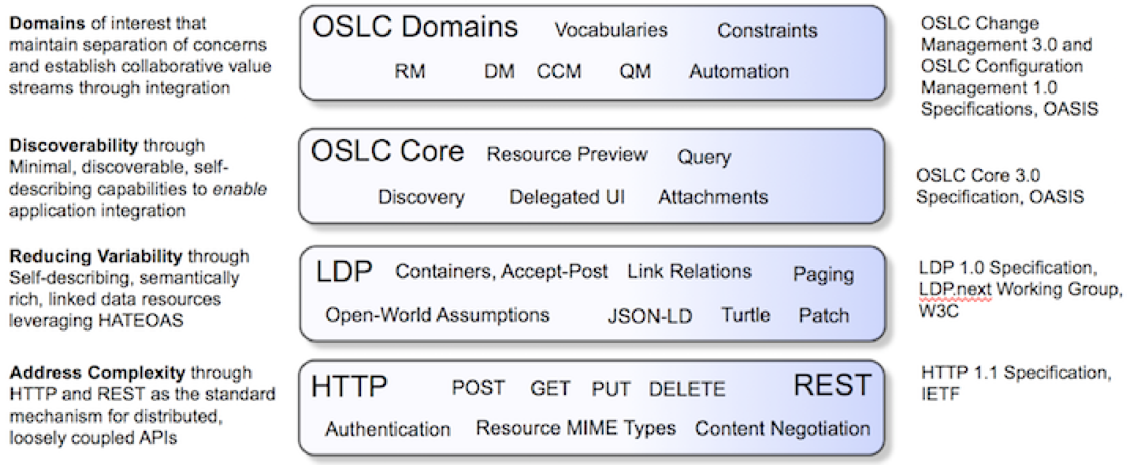

Domains of interest that maintain separation of concerns and establish collaborative value streams through integration
OSLC Domains
Vocabularies
Constraints
RM
DM
CCM
QM
Automation
OSLC Change Management 3.0 and OSLC Configuration Management 1.0 Specifications, OASIS
Discoverability through Minimal, discoverable, self-describing capabilities to enable application integration
OSLC Core
Resource Preview
Query
Discovery
Delegated UI
Attachments
OSLC Core 3.0 Specification, OASIS
Reducing Variability through Self-describing, semantically rich, linked data resources leveraging HATEOAS
LDP
Containers, Accept-Post
Link Relations
Paging
Open-World Assumptions
JSON-LD
Turtle
Patch
LDP 1.0 Specification, LDP.next Working Group, W3C
Address Complexity through HTTP and REST as the standard mechanism for distributed, loosely coupled APIs
HTTP
POST
GET
PUT
DELETE
REST
Authentication
Resource MIME Types
Content Negotiation
HTTP 1.1 Specification, IETF

Figure 3.1. *OSLC Architecture*

By utilizing the standards of the Semantic Web, OSLC defines its capabilities which are known as core capabilities. OSLC capabilities are developed on Semantic web principles [34] and allow integration of data with simple and best approach providing scalability, reusability, and traceability of resources. OSLC uses LDP to create, access, update and delete resources. LDP is a set of specifications to build RESTful services to expose

Linked Data resources. In LDP resources are exposed using standardized data formats such as RDF and these resources are identified and linked using URIs. OSLC provides a standard and consistent data model based on Resource Description Framework (RDF) to connect resources. OSLC interface consists of a set of RESTful services and can be implemented as an adaptor called an OSLC server. A tool vendor or a third-party plugin can provide the interface natively also.

3.2 OSLC Core Capabilities

3.2.1 Discovery

OSLC Discovery addresses one typical issue that happened in cross-domain interoperable systems, which is the availability of resources the client can avail. OSLC discovery provides a mechanism for the server to expose detailed information. This allows the client to determine the access and resources the server is exposing. In order to achieve this, OSLC defines a set of RESTful services and clauses to be fulfilled by an OSLC server[35]. Discovery allows clients to dynamically access resources based on the URLs provided in the response. This way the client can access all services by utilizing the links and information for seamless integration of tools or data interoperability.

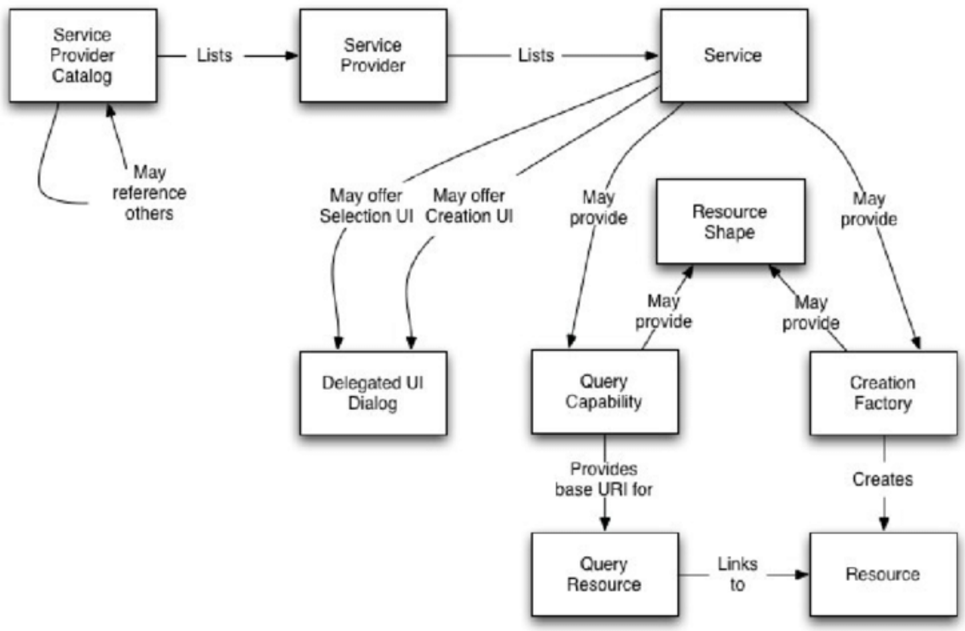


Figure 3.2. *OSLC Core Specification concept and relationships [35]*

Figure 3.2 present the OSLC core capabilities concept and relationships. According to this, discovery starts from the URL of the Service Provider Catalog.

A **OSLC Service Provider Catalog** refers to a collection of tools or services provided by an OSLC server. This is a very crucial component of the OSLC core as this act as starting point of the discovery mechanism. A Service Provider Catalog lists the details of all Service Providers attached to an OSLC server. The Service Provider Catalog respond with RESTful behaviour and allows clients to discover services provided by a server along with the links to access those providers.

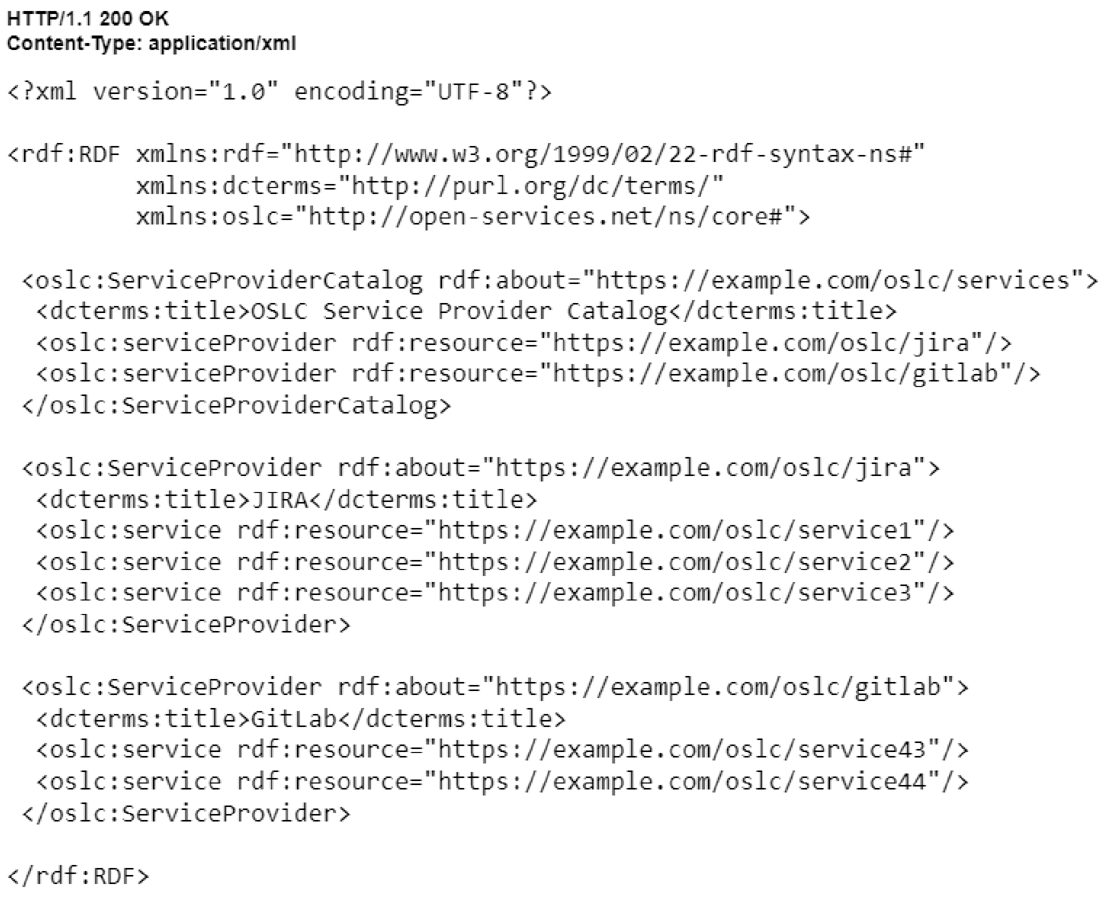
A **OSLC Service Provider** is a provider that offers some services. A Service Provider in a service provider catalog could be referred to as a single tool or multiple tools based on the use-case scenario. For example, an OSLC service specific to a domain like change management will have more than one software development tool that exposes resources related to change requests e.g. JIRA, GitLab, and Bugzilla. Each Service Provider further provides information on the services they offer along with the links to the services.

An **OSLC Service** defines resources in a standard way by exposing resources through REST API. The operations link with the resources is also exposed. For example, a Service Provider such as JIRA could have services related to change management like bugs, features, epics, and user stories. Each service will expose resources along with the links to access those resources. This will allow the client to explore resources within a service to create, update, and query resources.

These are the main concepts in OSLC core that further extend OSLC capabilities to operate in resources. OSLC defines a set of RESTful patterns for these above-defined concepts.

Discovery allows the client to further check on the access availability of a service using HTTP and resource shape. Figure 3.3 illustrates a simple example of OSLC Discovery. In this example, the client makes a request for a service provider catalog, acting as starting point for discovery. The client in response gets the detail of the service provider along with a list of service providers within the catalog. Furthermore, the response also gives detail of each service provider along with the services offered by that service provider. Each service provider or service will have URLs from which the client can access their desired service and get more details about a specific resource. Along with this OSLC discovery defines a set of vocabulary to be utilized by each component of the OSLC Core.

This makes OSLC Discovery the main mechanism of the OSLC Core concept. As demonstrated in 3.2, the Discovery follows the flow of the figure and exposes resources. Thus provides a set of specifications and clauses upon which an OSLC server can be created. So that an OSLC Client is able to see the available resources exposed by the server along with the URL of resources to access those further. the provided access to a service or resource and utilize it for integration purposes. Whereas, OSLC Discovery is not meant to discover something but provides a mechanism for the OSLC server to provide enough information about resources using REST architecture.



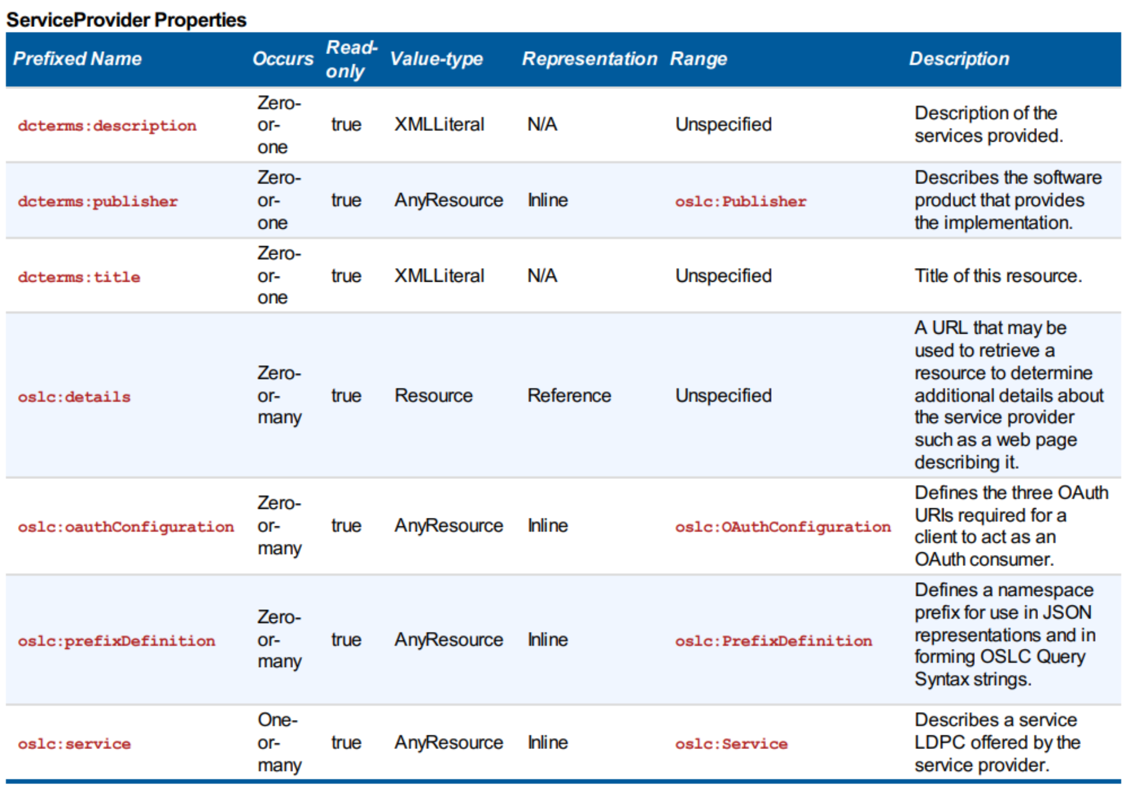
```
HTTP/1.1 200 OK
Content-Type: application/xml
<?xml version="1.0" encoding="UTF-8"?>

<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:dcterms="http://purl.org/dc/terms/"
         xmlns:oslc="http://open-services.net/ns/core#">

 <oslc:ServiceProviderCatalog rdf:about="https://example.com/oslc/services">
  <dcterms:title>OSLC Service Provider Catalog</dcterms:title>
  <oslc:serviceProvider rdf:resource="https://example.com/oslc/jira"/>
  <oslc:serviceProvider rdf:resource="https://example.com/oslc/gitlab"/>
 </oslc:ServiceProviderCatalog>

 <oslc:ServiceProvider rdf:about="https://example.com/oslc/jira">
  <dcterms:title>JIRA</dcterms:title>
  <oslc:service rdf:resource="https://example.com/oslc/service1"/>
  <oslc:service rdf:resource="https://example.com/oslc/service2"/>
  <oslc:service rdf:resource="https://example.com/oslc/service3"/>
 </oslc:ServiceProvider>

 <oslc:ServiceProvider rdf:about="https://example.com/oslc/gitlab">
  <dcterms:title>GitLab</dcterms:title>
  <oslc:service rdf:resource="https://example.com/oslc/service43"/>
  <oslc:service rdf:resource="https://example.com/oslc/service44"/>
 </oslc:ServiceProvider>

</rdf:RDF>
```

Figure 3.3. *OSLC Discovery example*

More, OSLC Discovery defines resource shape for each concept such as Service Provider Catalog or Service Provider. One example of these properties is shown in Figure 3.4, where general properties that can be applied to any service provider are described. As figure 3.3 demonstrates, a few properties are used to describe the resource such as dc-terms:title, oslc:service and others can also be used oslc:details. Each property has a description and constraints on the occurrence of resources. And an OSLC server should utilize them based on the needs while exposing resources upon a request from the client. Similar to Figure 3.4, OSLC core defines vocabularies for other concepts mentioned in Figure 3.2. Moreover, any OSLC server from any domain should follow the OSLC Core concept for exposing resources to the client [35]. This standardized approach allows an OSLC server to manage data from various domains in an efficient way.

3.2.2 Resource Preview

OSLC Resource Preview capability describes an approach to render the HTML representation of a resource. This capability defines a set of vocabulary which an OSLC server must implement for an OSLC client to preview resource [36]. This capability is useful for the client applications to render a preview of some resource URLs. Figure 3.5 illustrates



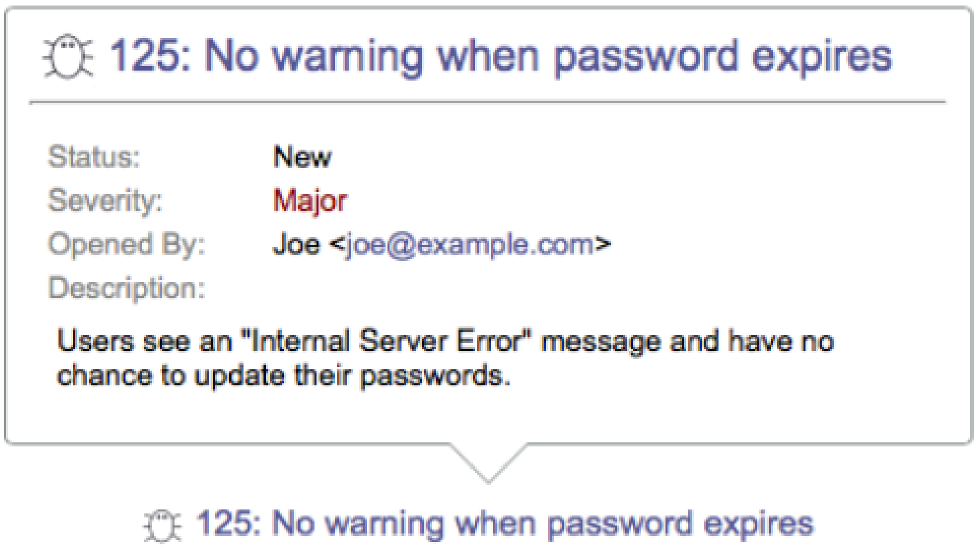
ServiceProvider Properties

Prefixed Name	Occurs	Read-only	Value-type	Representation	Range	Description
dcterms:description	Zero-or-one	true	XMLLiteral	N/A	Unspecified	Description of the services provided.
dcterms:publisher	Zero-or-one	true	AnyResource	Inline	oslc:Publisher	Describes the software product that provides the implementation.
dcterms:title	Zero-or-one	true	XMLLiteral	N/A	Unspecified	Title of this resource.
oslc:details	Zero-or-many	true	Resource	Reference	Unspecified	A URL that may be used to retrieve a resource to determine additional details about the service provider such as a web page describing it.
oslc:oauthConfiguration	Zero-or-many	true	AnyResource	Inline	oslc:OAuthConfiguration	Defines the three OAuth URIs required for a client to act as an OAuth consumer.
oslc:prefixDefinition	Zero-or-many	true	AnyResource	Inline	oslc:PrefixDefinition	Defines a namespace prefix for use in JSON representations and in forming OSLC Query Syntax strings.
oslc:service	One-or-many	true	AnyResource	Inline	oslc:Service	Describes a service LDPC offered by the service provider.

Figure 3.4. *ServiceProvider Properties [35]*

an example of this capability. It allows the client to preview additional information about a resource URL without opening another view. Therefore when resources of other tools are integrated with different tools the user will be able to have a preview of resources without leaving the current view. The client can hover over the resource URL and preview the resource. However, it might be the case that some resources don't allow previews due to security restrictions. OSLC defines a set of vocabulary [36] and specifications, such as server must return support certain media types such as application/json, text/turtle, application/x-oslc-compact+xml. In addition, the client must display the compact resource using an HTML iframe tag. Furthermore, this capability allows for adjustment of the size of the preview based on the user's choice. Only the URI of the resource is needed to preview the resource, which makes collaboration worthwhile and purposeful.

However, this capability is useful and more related to the case of tool integration where the user is able to see a preview of links to other resources with additional information without leaving their tool. Similar to Resource Preview there are some other OSLC capabilities that are more suitable for tool integration rather than developing data adapters. Although the capability provides a set of rules for adapters also, however, these rules are meant for OSLC clients. Hence the adapter is designed in a way that the OSLC client is able to integrate tools in a toolchain. Capabilities such as Delegated Dialogs, and Creation Factory belong to this category.



125: No warning when password expires

Status: New
Severity: Major
Opened By: Joe <joe@example.com>
Description:

Users see an "Internal Server Error" message and have no chance to update their passwords.

125: No warning when password expires

Figure 3.5. *Preview of a link [36]*

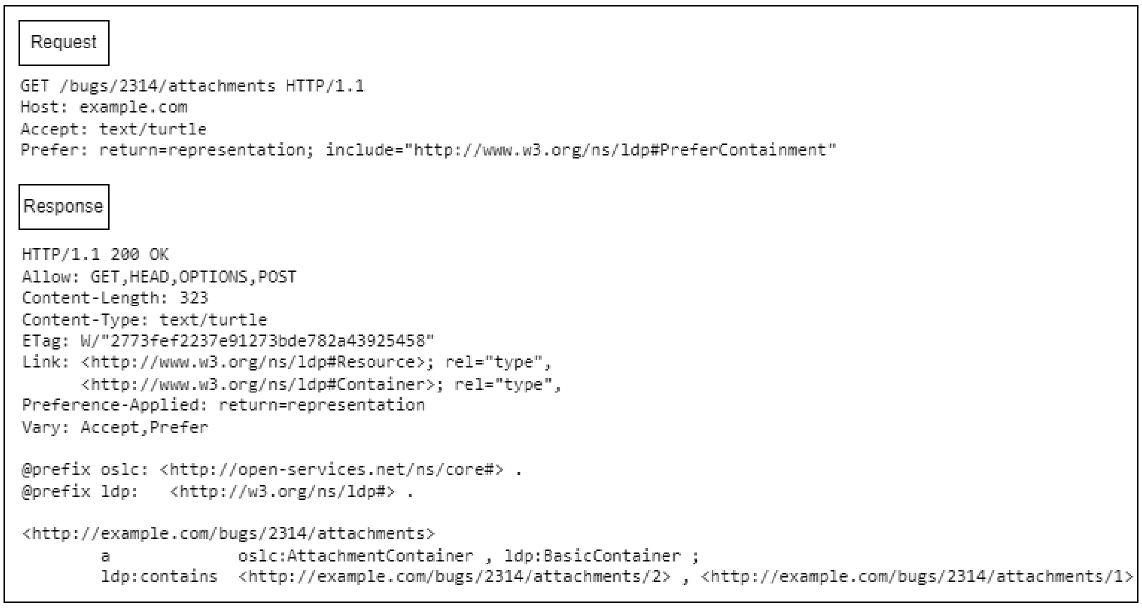
Delegated Dialogs

This OSLC capability serves the client with an embedded UI for the creation or selection of resources for other applications. This allows users to create or select resources with a UI interface rather than an HTTP interface. Further details of this capability can be found here [37].

3.2.3 Attachments

OSLC specify an approach to attach documents in reference to other resources. These attachments can be associated with an RDF resource or a non-RDF resource [38]. This capability can be useful in a situation such as creating a task and then attaching a log file to summarize the problem or attachment of designs or screenshots to a specific user story. OSLC attachment capability describes a way to manage attachments linked with resources. This capability utilizes the LDP[48] model and allows creating, updating, deletion and relation access of attachments. The client can also check that either the resource supports attaching attachments.

Figure 3.6 illustrates a request and response example. In this example, a GET request is made for fetching attachments of a bug where Prefer Header is used as described in the OSLC Attachments specification. The OSLC server responded with a list of attachments containerized in ldp:contains part related to the requested bug. The response container is an oslc:AttachmentContainer using LDP [48] Basic Container type. Both requests and responses are made according to the set of specifications defined in OSLC Attachments. An OSLC-compliant server desiring to utilize this capability must implement the set of the



Request

```
GET /bugs/2314/attachments HTTP/1.1
Host: example.com
Accept: text/turtle
Prefer: return=representation; include="http://www.w3.org/ns/ldp#PreferContainment"
```

Response

```
HTTP/1.1 200 OK
Allow: GET,HEAD,OPTIONS,POST
Content-Length: 323
Content-Type: text/turtle
ETag: W/"2773fef2237e91273bde782a43925458"
Link: <http://www.w3.org/ns/ldp#Resource>; rel="type",
      <http://www.w3.org/ns/ldp#Container>; rel="type",
Preference-Applied: return=representation
Vary: Accept,Prefer

@prefix oslc: <http://open-services.net/ns/core#> .
@prefix ldp:   <http://w3.org/ns/ldp#> .

<http://example.com/bugs/2314/attachments>
        a              oslc:AttachmentContainer , ldp:BasicContainer ;
        ldp:contains   <http://example.com/bugs/2314/attachments/2> , <http://example.com/bugs/2314/attachments/1>
```

Figure 3.6. *Example of attachment capability [38]*

vocabulary defined in the document of OSLC Attachments [38].

3.2.4 Resource Shapes

In OSLC Resource shapes are integrity constraints that each resource should satisfy [39]. In RDF resources are described as triplets and each triplet has a set of representations called shape. These shapes act as a validation of data means the data will be only correct if it satisfies the described shape for a certain resource, instance or property. Each entity of a triplet has some shape that describes constraints on that entity. SHACL[28] is one example of RDF data validation. These constraints are required if the data is exposed to some API and the API documentation will allow the API consumer to know the shape of the resource. This resource shape allows the user to know the data set so that the user can write query data easily. Hence resource shape is a set of constraints to any entity that exists in an RDF triplet. An OSLC-compliant server must satisfy the shape constraints described in OSLC vocabulary. For example, in Figure 3.4, if we consider dcterm:title property, the constraints on this property are: Occurs (Zero or one), Readonly (true), Value-type (XMLLiteral). These entities for this property present content that dcterm:title property must follow. For example, if dcterm:title has two values then the process will be interpreted as an error because the constraint represents a zero-one value.

Figure 3.7 explains this concept: a resource has some property, and then the property can have constraints about the content of that property. These constraints are called shape, which means what a resource should have and how it will look alike.

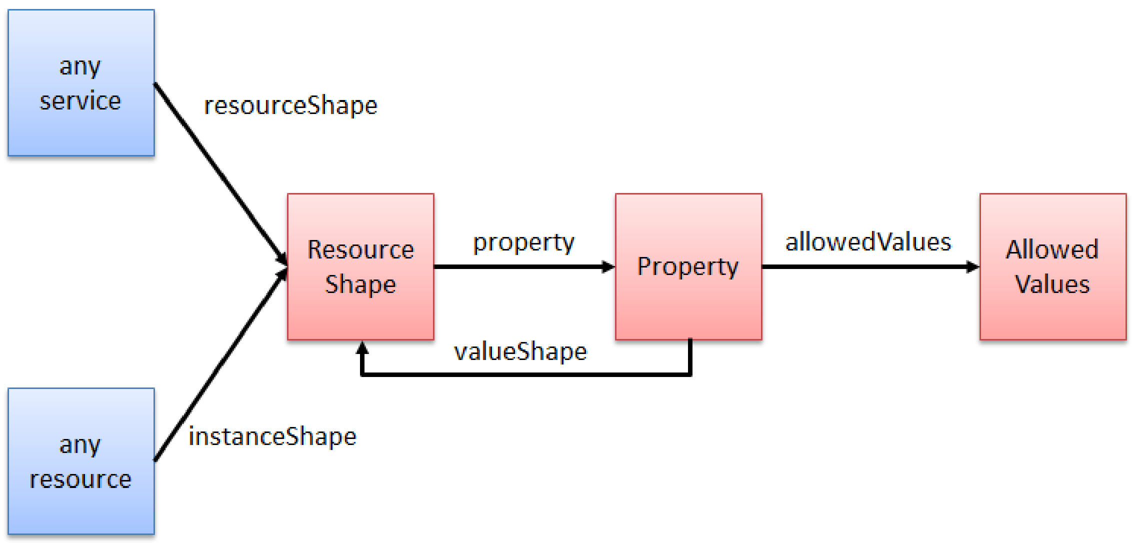


Figure 3.7. *Main concept of resource shape [39]*

3.2.5 Tracked Resource Set

OSLC Tracked Resource Set (TRS) capability is a set of specifications that allows a server to expose resources in a way that an OSLC client is able to keep track of a resource or a set of resources [42]. A server providing TRS capability is referred to as a TRS server in OSLC. In the software lifecycle, some resources go under continuous change. Therefore, the OSLC client wants to track these kinds of resources. OSLC TRS allows an OSLC server to expose resources and the data linked data related to that resource in a way. So that the OSLC client is able to maintain, live searchable information along with the change history. In OSLC TRS specification, a TRS resource is categorized into two blocks: the base and the change log. The base is presented as an LDP container having the latest information about the tracked resource. However, the change log holds the history of changes occurring such as additions, and deletions to that resource. Therefore for the applicability of OSLC TRS, a server must be conformant to vocabularies and clauses defined in TRS specifications [42]. TRS can be beneficial for a set of big resources that undergoes continuous change so that the client is permitted to aggregate, build and maintain information.

3.2.6 Authentication

Authentication is crucial in a collaboration environment, where providers are given access to resources based on user privilege. Authentication will allow the discovery of the accessibility of resources. OSLC doesn't provide its own authentication mechanism but endorses standard web protocol for authentication purposes. According to the OSLC Authentication clause, the OSLC server must protect resources with some authentication method. OSLC server may use HTTP basic authentication with SSL (Secure Socket

Layer) only. For secure association with resource providers, OSLC recommends OAuth 2.0 [11] along with OpenID Connect [30] that runs on top of OAuth 2.0 Framework [34]. This means while creating an OSLC server, some authentication method must be utilized to expose resources for the consumer.

3.2.7 Query

OSLC defines its own query capability to query data. OSLC query allows to search for RDF resources and return response body in RDF representation. OSLC query capability is analogous to SPARQL, however, it has minimal support for querying data such as matching, filtering or ordering [41]. OSLC servers are mostly adaptors constructed of existing data sets that don't follow the RDF pattern, therefore do not provide SPARQL endpoints to query data. Thus, OSLC query capability is simple enough to support a broad scope of server architectures and endurance technologies to query. Hence, this makes OSLC query to be applicable to a wide range of information from various vendors. To provide query features an OSLC service must implement endpoints using OSLC query specifications with RESTful pattern [41]. The OSLC query capability provides these parameters: oslc.where, oslc.select, oslc.orderBy, oslc.prefix, oslc.searchTerms, oslc.paging, oslc.pageSize. These parameters are general enough to meet basic query needs in data interoperability. An OSLC server must be conformant to clauses and a set of rules defined in OSLC query capability specification.

3.2.8 Resource Operation

OSLC determines that the servers must adhere to HTTP [15] specifications while performing create, read, update and delete (C.R.U.D) operations on the resources. This means that the server should respond with the resources along with the operations that a client can perform on the resource. For example, other than HTTP's popular methods such as GET, POST, PUT, and DELETE, the server should support GET requests with query parameters to search a specific set of resources, the server should support GET requests with version-specific URLs of resources. OSLC Core describe a set of HTTP REST services concerning loosely coupled and flexible tool integration.

3.3 Domain Specifications

OSLC has defined several domain specifications that extend core capabilities and can be utilized for collaboration as per the tool use case. All the domains specify their own vocabularies that support their use case for common integration support. OSLC domains are actively maintained on OSLC Open Project Specification [40]. Some OSLC domains are described in figure 3.8.

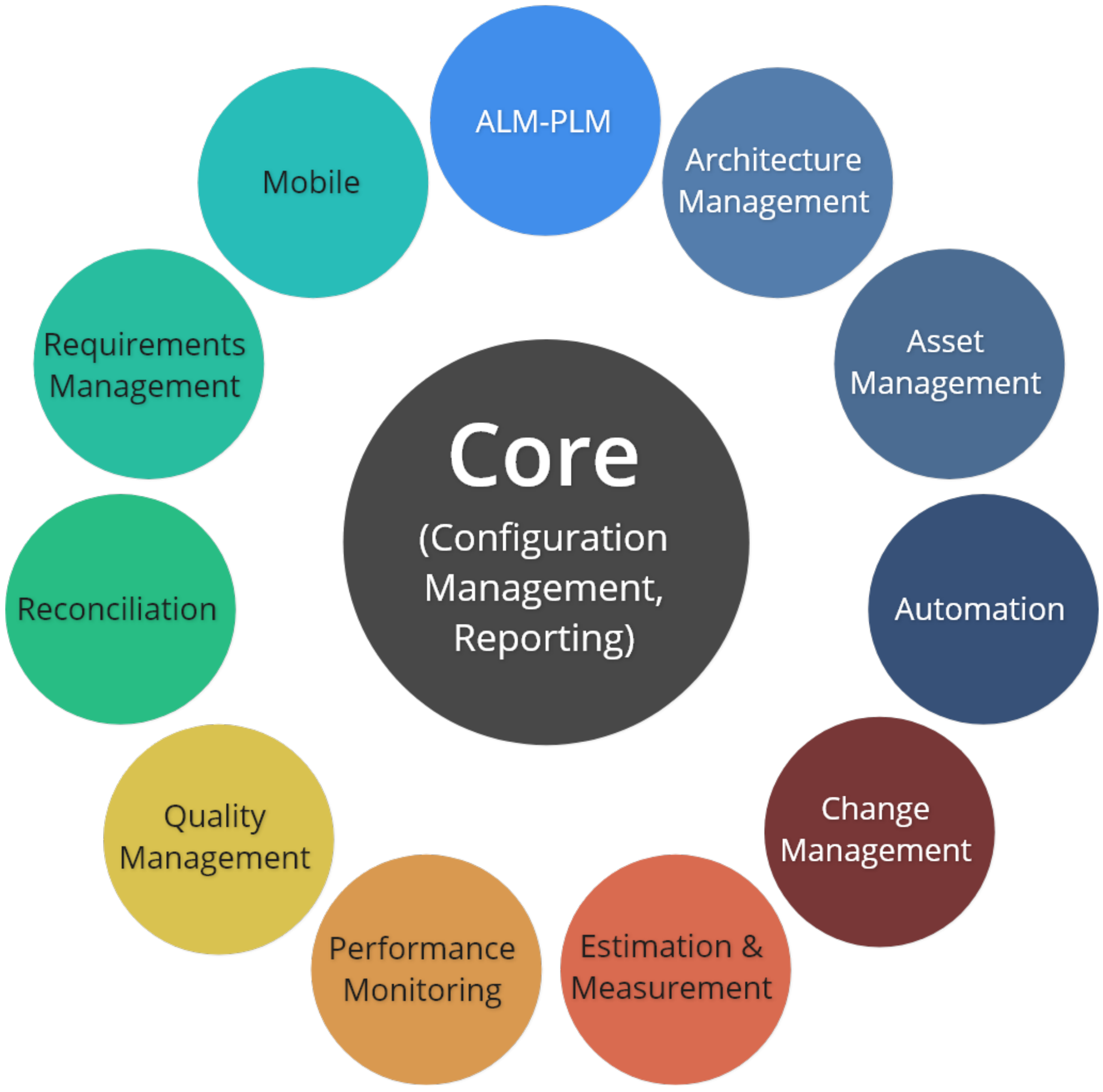


Figure 3.8. *OSLC Domains [1]*

3.3.1 Change Management

OSLC Change Management domain specifications declare a set of RESTful interfaces for HTTP methods such as POST, GET, PUT, and DELETE and content type handling, response codes and resource formats. These interfaces can be utilized for the management of product change requests, activities, tasks and relationships between those and related resources such as project, category, release and plan. A server built on a Change management domain can be called a CM server. CM domain declares some extended vocabularies [32] that support resources of its type for collaboration. The RESTful interfaces are not specific to any domain but follow the OSLC Core specifications. For example, OSLC CM should provide authentication, error responses and representation of resources such as turtle, XML, HTML, and JSON, which are defined on OSLC Core. However, the vocabularies are specific to each domain and allow a domain to handle data from various tools of a similar kind. The vocabularies defined for the OSLC CM domain are general enough to be applied to data related to a change request regardless of the tool such as JIRA, GitLab, Trello, or Bugzilla. An OSLC CM server must adhere to clauses and constraints defined in OSLC CM specifications [31]. An overview of OSLC CM vocabularies and this relationship can be seen in figure 3.9.

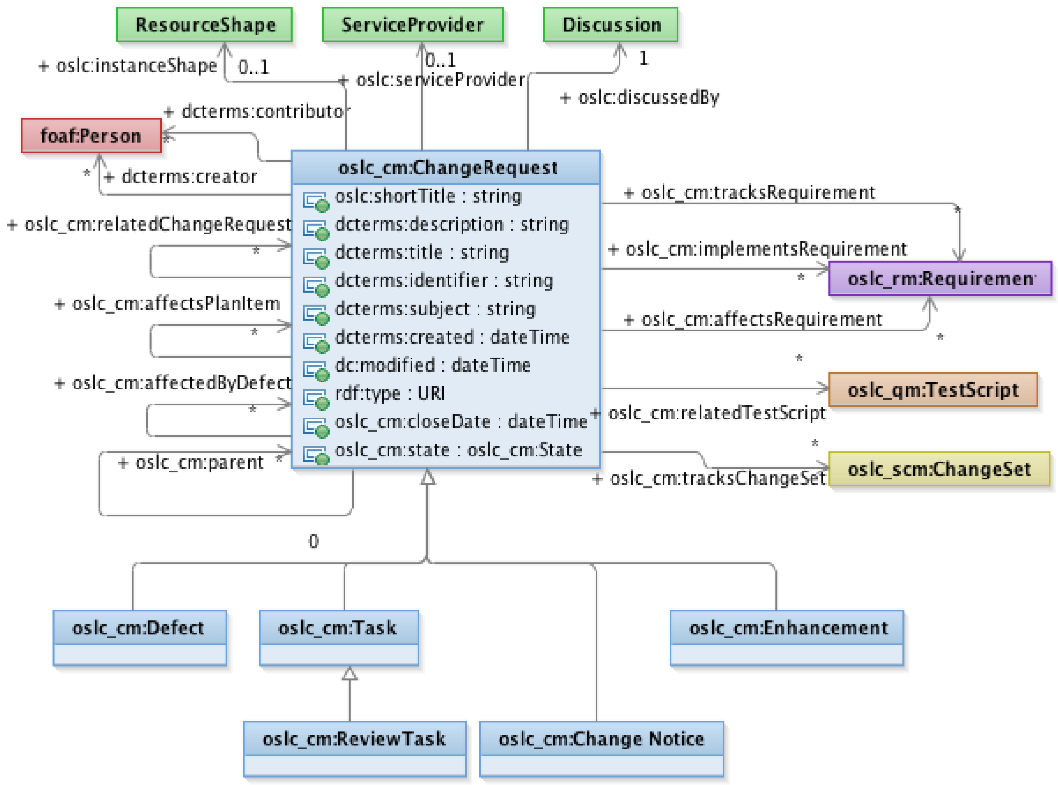


Figure 3.9. *OSLC Change Management Vocabulary [32]*

Explaining OSLC CM regarding general requirements and RESTful interface, table 3.1 explain some requirements that an OSLC-compliant CM server must fulfil. Most of the requirements are from OSLC Core specifications and some are domain-specific requirements such as query capabilities, turtle representations, RDF/XML representations, and JSON representations.

3.3.2 Other Domains

Configuration Management This OSLC domain specifies a set of REST APIs and RDF vocabulary for managing versions and configurations of linked data resources for multiple domains. It defines a set of methods for creating, deletion and updating configured resources and also provides a way of differentiating a versioned resource for configuration purposes. Further information on OSLC Configuration Management, its vocabularies and other constraints can be obtained from here [33].

Table 3.1. *OSLC CM requirements [31]*

Requirements	**Explanation**
Unknown properties and content	OSLC servers **MAY** ignore unknown content.
Resource Operations	OSLC service **MUST** support resource operations via standard HTTP operations.
Resource Paging	OSLC servers **MAY** provide paging for resources but only when specifically requested by a client.

Requirements Management OSLC Requirement Management domain defines vocabulary and RESTful methods for resources concerning Requirement management like requirements, requirements collections and supporting resources defined in OSLC core specification. This domain can be called OSLC RM.

Quality Management OSLC Quality Management domain provides a set of RESTful HTTP methods like GET, POST, UPDATE, DELETE and vocabularies for resources like test cases, test plans, and test results of a software delivery lifecycle.

Similar to these, OSLC defines several other domains related to Application Lifecycle Management (ALM), and DevOps. As per OSLC community discussions, one can add a new domain if it doesn't exist in OSLC Domains documentation by following OSLC core specifications.

4. RESEARCH METHODOLOGY

This chapter briefly explains the context of the research upon which the investigation was based and the approach used to conduct the research. This research was taken in an iterative way where having a weekly feedback session. Qualitative research was performed mainly by analysing OSLC specification documents, literature review, and joining the OSLC community group.

4.1 Context of Research

In DevOps, the development and operations team collaborates to speed up the delivery process while ensuring a quality product. This process involves several tools at various stages of development and delivery. These tools are having different vendors and are thus not meant to interoperate, due to which the toolchain is not connected. In addition, the managers lack traceability of the whole situation and developers only have information about the technical side, not the whole picture of the project. Similarly to this other stakeholders involved cannot get the overall status and are unaware of the actual happening in the project. This leads the stakeholders to make decisions based on their experience but not against the situation. Therefore these misconceptions lead to project delay or failure [10].

VISDOM's objective is to follow the above-mentioned issues and provide a data visualisation of the project. The major components of VISDOM's data management systems are data fetchers and data adaptors.

Data fetchers are responsible for fetching data from data sources which could be development tools, databases, or repositories utilized in a software project. In addition, fetchers are saving the accumulated data to a storage system. The data is in raw shape and along with the raw data saving metadata. The second and most crucial component of the data management system is a data adapter.

The **data adapter** is responsible for further filtering the data and making it visualization ready. However, the reason is that data sources with different vendors may end with different data formats. Therefore data interoperability becomes very complex and it's difficult to filter data for one view.

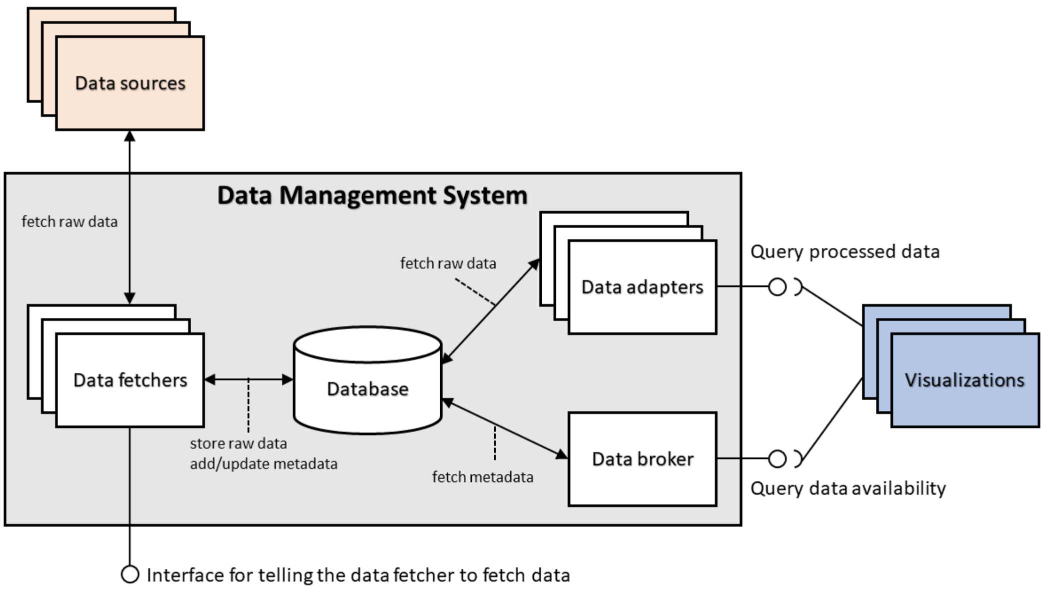


Figure 4.1. *VISDOM data management system [50]*

Figure 4.1 illustrates the data management system of the VISDOM. Data adapters are accepting raw data and provide unified data for visualization after required processing. In addition to this, data adapters are also responsible for providing additional information and linking to the data if it is not available in the raw data. So that the user can further dive into the data and get more information on the matter. However, the visualization dashboard varies on stakeholders so the data adapter will process data based on the stakeholder request. As a result of this, every request will require different data. Thus, this requirement makes adapters the most complex components of the VISDOM data management system and is the main topic of this thesis. This study will provide the possible solutions that OSLC can provide, to standardize the VISDOM data adapter also refers as a data model.

4.2 Research Problem

The main focus of this thesis is the data adapter component of the VISDOM data management system illustrated in Figure 4.1. VISDOM has been tested earlier in different scenarios where data management system architecture was customized for each scenario. Each pilot implementation scenario was different based on the data sources, implementation tools, storage system and situation. Below mentioned factors were highlighted:

- Re-usability of modules in data adapter component
- Creation of APIs that provide support for some query language
- Ability to add new data sources dynamically
- Security of data is crucial

The current data adapter component has modules for each data source which are not reusable in the sense that one module can be utilized for another data source of a similar kind. For example, data sources from Version management tools such as GitHub, GitLab, or BitBucket have similar kinds of data such as repositories, commits, and branches. This means that a module created for GitHub can be reused for other data sources of a similar kind like GitLab, BitBucket, and CodeCommit.

Moreover, when the number of requests from the visualization dashboard increases, the data processing becomes slow. Therefore, the API endpoint should provide some query language so that only the required data is processed.

The data adapters should have the ability to provide data dynamically. This means that data adaptors may be able to provide different data based on the types of visualization. As the VISDOM provide stakeholders-based visualisation such as a different dashboard for admins, managers, and developers. Therefore same data adaptor will be serving different data to different visualisation dashboards. This means that the data adaptor should be able to provide dynamic data upon various requests.

The last factor is the data should be secured using some authentication. As data privacy is important, some strong way of authentication is required to make sure that the data is only fetched by a user having access rights. For example, the data adaptor should have some authentication acting as a middleware on each request to verify the user type such as admin or simple user. And the adaptor will visualize the only data based on the user's access category.

These factors are considered while doing the research.

4.3 Iterative Research

The research was carried out in an iterative way where a meeting was held with the VISDOM team each week. The process goes such a way that each week current findings of OSLC were represented to get feedback. The investigation was mainly based on the OSLC documentation analysis, upon which feedback sessions were carried out.

The descriptive investigation technique was adopted to complete the study to search for answers to the research questions. A thorough research was done on the gathered data to understand the situation and then focus on problems described initially in the introduction.

This approach was a good fit for this research as the scope of a broad view was to investigate the OSLC standard and analyze the VISDOM project based on its needs. Therefore in this context, descriptive investigation fits nicely along with the document analysis of both sides, OSLC and the VISDOM.

4.4 Data Collection

The data collection was done through the analysis of OSLC documentation. These documents include OSLC specification documents, OSLC implementation guidelines and requirements and case studies implementing OSLC. Some data was gathered by questioning OSLC experts and the maintainer's community. This community comprise experts from different areas of software engineering and is currently contributing to the standard. In addition to this, data related to the VISDOM project was gathered by reviewing VISDOM documentation and questioning the VISDOM team during feedback sessions.

5. FINDINGS AND DISCUSSION

This chapter will present the findings of this study research. A summary of OSLC offerings will be explained along with the benefits achieved by implementing OSLC for data interoperability in a cross-domain scenario. One part of this chapter will derive collected data from the study and the other part describe and analyze the data with interpretation based on the research problem outlined in the previous chapter.

5.1 Description

The analysis of OSLC data has revealed the below findings based on the research problem.

1. OSLC, an open standard based on semantic web stack and W3C standards provide standardized solutions for tool integration and data interoperability in the software development life-cycle

2. OSLC utilizes a linked data approach for linking data of any kind. The linked data is represented in the RDF graph, the formation of a standard triple linked with a subject, a predicate and an object. OSLC adopt LDP specifications to expose linked data with RESTful web services.

3. As described before, LDP is a set of specifications allowing linked data to be exposed on the web using HTTP protocol and RESTful services.

4. On top of LDP specifications, OSLC defines its own rules and clauses to provide a RESTful pattern for the interoperability of heterogeneous data.

5. By the means of utilizing semantics and forming linked data allows the OSLC complaint data model to connect and manage complex data.

6. In addition, OSLC defines its own query capability to have query endpoints, this capability is based on SPARQL a query language designed for RDF data models.

8. OSLC implementation be done with Lyo Designer[26], an Eclipse plugin that allows the creation of overall system architecture graphically. Lyo Designer utilizes the OSLC4J library and upon creation of the information model for resources and each server operations and service, it generates an initial skeleton of classes in Java.

5.2 Analysis

This section will demonstrate the findings in more detail.

OSLC adopts W3C standards and provides a concept of making resources available. Using LDP specifications, OSLC allows a server to group resources in containers. This approach eases the procedure of discovering resources by calling the container and linking resources from one container to another. The RESTful architecture helps the OSLC server to handle request and response processes in a standard manner. The concepts of having stages in OSLC such as Service Provider Catalog, where all the service providers are listed. A single OSLC service provider and Services that expose methods allowed to be performed on a selected resource. This OSLC Core concept provides a mechanism to expose resources on the web and allows data interoperability in a standardized way. Therefore, OSLC servers can manage complex data sources and provide ease of adding new data sources.

Furthermore, OSLC domains utilise the Core concept and define a set of specifications and vocabularies to allow the creation of domain-specific servers with a standardised approach. Whereas, this helps to create and manage systems that involve a variety of domains within the software development lifecycle. OSLC domains approach allows complex systems to be more scalable due to independent domains. This further makes it easier to integrate more data sources with different domains within the system. As per the OSLC community group, OSLC allows one to create a new domain that is not provided yet in the OSLC domains system. This can be done by following OSLC Core concepts and defining resource shape constraints to that specific domain resources.

Lyo Designer is a plugin that provides ease for OSLC implementation. It allows the creation of system architecture graphically, where various artefacts can be defined in the model. Based on the model created, it creates classes and objects as base code for further implementation [26]. This eases the development process for the developers, so they can implement the functionality in the classes.

In summary, the OSLC-based data adaptor approach will have the following benefits:

- Having a well-established REST architecture following HTTP protocol.
- Eases interoperability of data collected from heterogeneous data sources.
- Server has a standard data representation with RDF, that eases the linking process.
- Increases reusability of modules as all the domains are based on OSLC Core.
- If modules are reused, the development cost is relatively low.
- OSLC domains allow loosely coupled one-to-many linking rather than one-to-one.

Some possible limitations drawn from the study are as follows:

- OSLC-based architecture is sensitive to URIs, so it must be stable to avoid broken links: As OSLC uses URI to represent every entity in a triplet, the server must use stable URIs to avoid any interruption or error.
- The development cost for VISDOM to have an OSLC-based architecture might be higher: Although OSLC allows you to build a server with any programming language of your choice [1], however, the mature version of OSLC has complete support in JAVA. This is one factor that will increase the cost for VISDOM as currently VISDOM is developed on Javascript. OSLC server will be completely different from the current VISDOM implementation which requires more time to build a new server.
- Some of the technologies in OSLC are new such as RDF which might end up with limited technology support.

OSLC can provide a standard way for VISDOM to expose resources of various data sources. A couple of scenarios are described based on the analysis. OSLC will allow VISDOM to create tool-specific or domain-specific servers. The first example will present a tool-specific scenario, where GitLab is taken as an example.

GitLab [21] is a software development tool that offers several services for the software development process such as version management (repositories), change management (tickets, boards, milestones), automation (pipelines) and maybe more services. An OSLC server can ease the process of defining all the services such as in the Service Provider Catalog GitLab will be expressed as Service Provider. The services offered by this tool will fall into the services category. As the services belong to a different domain, each service will implement its respective OSLC domain. The server should support OSLC query capability for each service. As described in OSLC domains specification, oslc:domain property can either be applied to an oslc:service or oslc:ServiceProviderCatalog [31].

Furthermore, each domain can have resources that fall into different categories for example, in change management service there could be resources related to tickets, milestones, and epics. Utilizing LDP containers, OSLC allows containerizing resources of a similar kind for easing the process of discovery for the client. This means that upon request to a container, in response the client will get the list of all the resources belonging to that container. This is one approach that provides flexibility and eases exposing information for better interoperability for VISDOM.

The second scenario is to create a domain-specific server where a single domain can have multiple oslc:ServiceProvider of a similar kind. Let's look at an example scenario where Change Management an oslc:domain act as a oslc:ServiceProviderCatalog and contains various oslc:ServiceProviders as software development tools such as JIRA, Bugzilla, GitLab. Each service provider is exposing resources related to the change management domain. Whereas, LDP container allows the creation of containers having similar resources from all the services. OSLC eases the process of unifying data from different tools of the same nature. This decreases the development process for creating separate servers for each tool.

In both examples, OSLC provides a standard way to expose resources and ease the interoperability process for VISDOM. As OSLC is sensitive to URI, so the information should be clean of broken links.

OSLC-based adaptors can have the ability to respond in dynamic behaviour, for example, based on the research problem each visualisation can request different data. OSLC specification eases the linking of data on semantic basis and provides query language to further filter the data. This ability allows the OSLC adaptor to respond with dynamic behaviour.

Furthermore, OSLC requires an OSLC-compliant server to implement some authentication method. The various authentication methods that OSLC support are described in Section 3.2.6. For OSLC servers, authentication is an essential aspect to build secure OSLC services. To ensure data privacy, OSLC recommends using OAuth, as it is widely used and provides a secure way to authenticate.

5.3 Architecture Proposal

In this section, a high-level architectural for VISDOM is proposed regarding OSLC implementation by following these works of literature [51] [8] [29] and own thoughts about OSLC are presented.

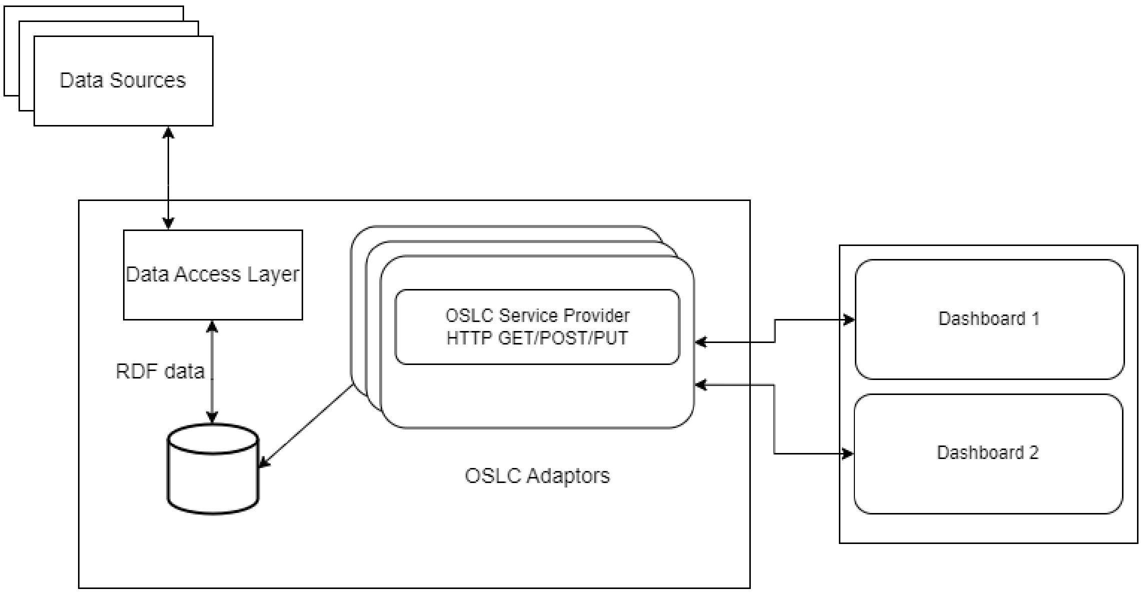


Figure 5.1. *VISDOM architecture with OSLC implementation*

The example scenario explanation compliments this architecture. In this architecture, there will be layered components starting from the bottom of figure 5.1:

Data Source: Data sources layer could be various tools, or databases that hold data from various domains. These will be the origin of the data.

Data Access Layer: This layer will fetch data from sources by calling their provided APIs and provide the data to the data adaptor in the RDF representation. This layer could have similar responsibilities to the Data Fetcher component of the VISDOMs Data Management system, as described in Figure 4.1. Whereas, this layer will be responsible for providing data to the data adaptor in RDF form. This will act as an extraction layer between the data source and the data adaptor layer.

OSLC Data adaptor: This will be the main component of this architecture where OSLC Core specifications are applied. Based on OSLC resource shape specification, the adaptor will implement the resource constraints and structure by utilizing the RDF schema. The adaptor will provide OSLC-compliant data to the Service Provider. This layer will be responsible for the implementation of the Service Provider Catalog and executing the OSLC Discovery concept. Furthermore, domain-specific resource constraints and clauses will also be implemented in this layer.

OSLC Service Provider: This layer will implement RESTful architecture for OSLC-based resources and expose resources based on any domain of its kind. It will make resources available for the client to be accessed by HTTP protocol methods. The linking of data can be done at this point. This layer can also provide query endpoints by utilizing OSLC query capability.

In figure 5.1, the data process block contains multiple OSLC adaptors providing visualization-ready data to the dashboard view. In terms of performance, the data converted by the data access layer can be saved in a database that supports RDF data and the data adaptor can retrieve the data from the database.

6. CONCLUSION

This research studied the VISDOM project, a visualization platform for DevOps software development. The VISDOM project has separate functionality. One of the VISDOM main blocks is its data management system, as VISDOM deals with various data sources that are heterogeneous in nature. The purpose of the data management system is to process and deliver visualization-ready data to the front-end side. The data adaptor is the most crucial and complex part of the VISDOM data management system as this component is responsible for providing visualization-ready data and linking data. Currently, the data adaptor has several challenges in processing data such as the reusability of modules for a data source of a similar domain, linking of data, and query endpoints for specific searches.

This study performed a thorough investigation of the OSLC standard that provides a standardised approach for tool integration and data interoperability for the development lifecycle. OSLC allows servers to expose resources and link data without any dependencies, as the domains are independent and can link with each other.

As OSLC describe independent domains, this avoid the system to be complex and allows for adding more data sources. OSLC capabilities are based on integration and interoperability purpose, so all the specifications are defined to be general for any data source. OSLC defines a set of instructions along with vocabularies and constraints on resources for each capability and domain. OSLC has no technology restriction for implementing, OSLC-compliant servers, however, it provides more support with JAVA. OSLC standard is the only standard providing solutions for tool integration and data interoperability at lifecycle level [4].

OSLC can benefit the VISDOM data adaptor, however, the reference architecture of VISDOM might be affected. OSLC will allow VISDOM to add and manage various data sources in an efficient way. The domain-driven behaviour of OSLC will allow VISDOM to have servers that can be reused for other data sources of the same domain. Whereas, few literature on OSLC might create blockers for OSLC implementation on VISDOM and can be costly in the end.

REFERENCES

[1] *About the OSLC Open Project*. URL: https://open-services.net/ (visited on 11/2022).

[2] Renzo Angles and Claudio Gutierrez. “The expressive power of SPARQL”. In: *International Semantic Web Conference*. Springer. 2008, pp. 114–129.

[3] David Beckett et al. “RDF 1.1 Turtle”. In: *World Wide Web Consortium* (2014), pp. 18–31.

[4] Olivier Berger et al. “Introducing OSLC, an open standard for interoperability of open source development tools”. In: *ICSSEA 2011*. 2011, ISSN–0295.

[5] Tim Berners-Lee. *Linked Data*. July 2006. URL: https://www.w3.org/DesignIssues/LinkedData.html (visited on 08/2022).

[6] Tim Berners-Lee, Roy Fielding, and Larry Masinter. *Uniform resource identifier (URI): Generic syntax*. Tech. rep. 2005.

[7] Tim Berners-Lee, James Hendler, and Ora Lassila. “The semantic web”. In: *Scientific american* 284.5 (2001), pp. 34–43.

[8] Matthias Biehl, Jad El-Khoury, and Martin Törngren. “High-level specification and code generation for service-oriented tool adapters”. In: *2012 12th International Conference on Computational Science and Its Applications*. IEEE. 2012, pp. 35–42.

[9] Chris Bizer, Richard Cyganiak, Tom Heath, et al. “How to publish linked data on the web”. In: (2007).

[10] Marcelo Cataldo and James D Herbsleb. “Coordination breakdowns and their impact on development productivity and software failures”. In: *IEEE Transactions on Software Engineering* 39.3 (2012), pp. 343–360.

[11] Ed. D. Hardt. *The OAuth 2.0 Authorization Framework*. Oct. 2012. URL: https://www.rfc-editor.org/rfc/rfc6749 (visited on 07/2022).

[12] *Dublin Core*. URL: https://www.dublincore.org/specifications/dublin-core/dcmi-terms/ (visited on 08/2022).

[13] Bob DuCharme. *Learning SPARQL: querying and updating with SPARQL 1.1*. " O'Reilly Media, Inc.", 2013.

[14] Roy Fielding. *Representational State Transfer*. URL: https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm (visited on 10/2022).

[15] Roy Fielding and Julian Reschke. *Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing*. June 2014. URL: https://httpwg.org/specs/rfc7230.html (visited on 07/2022).

[16] Roy T Fielding et al. “Reflections on the REST architectural style and" principled design of the modern web architecture"(impact paper award)”. In: *Proceedings of the 2017 11th joint meeting on foundations of software engineering*. 2017, pp. 4–14.

[17] Roy Thomas Fielding. *Architectural styles and the design of network-based software architectures*. University of California, Irvine, 2000.

[18] *FOAF Vocabulary Specification*. URL: http://xmlns.com/foaf/0.1/ (visited on 08/2022).

[19] Ned Freed and Nathaniel Borenstein. “Multipurpose internet mail extensions (MIME) part one: Format of internet message bodies”. In: (1996).

[20] Fabien Gandon and A Th Schreiber. “Rdf 1.1 xml syntax”. In: (2014).

[21] *GitLab*. URL: https://about.gitlab.com/ (visited on 05/2023).

[22] Steve Harris and Andy Seaborne, eds. *SPARQL 1.1 Query Language*. URL: https://www.w3.org/TR/sparql11-query/ (visited on 10/2022).

[23] Pascal Hitzler et al. “OWL 2 web ontology language primer”. In: *W3C recommendation* 27.1 (2009), p. 123.

[24] A Clara Kanmani, T Chockalingam, and N Guruprasad. “RDF data model and its multi reification approaches: A comprehensive comparitive analysis”. In: *2016 International Conference on Inventive Computation Technologies (ICICT)*. Vol. 1. IEEE. 2016, pp. 1–5.

[25] Gregg Kellogg, Pierre-Antoine Champin, and Dave Longley. “Json-ld 1.1–a json-based serialization for linked data”. PhD thesis. W3C, 2019.

[26] Jad El-khoury. “Lyo code generator: A model-based code generator for the development of oslc-compliant tool interfaces”. In: *SoftwareX* 5 (2016), pp. 190–194.

[27] Graham Klyne. “Resource description framework (RDF): Concepts and abstract syntax”. In: *http://www. w3. org/TR/2004/REC-rdf-concepts-20040210/* (2004).

[28] Holger Knublauch and Dimitris Kontokostas. “Shapes constraint language (SHACL)”. In: *W3C Candidate Recommendation* 11.8 (2017).

[29] Luka Lednicki et al. “Integrating version control in a standardized service-oriented tool chain”. In: *2016 IEEE 40th Annual Computer Software and Applications Conference (COMPSAC)*. Vol. 1. IEEE. 2016, pp. 323–328.

[30] *OpenID Connect*. URL: https://openid.net/connect/ (visited on 06/2022).

[31] *OSLC Change Management Version 3.0. Part 1: Specification*. May 2021. URL: https://docs.oasis-open-projects.org/oslc-op/cm/v3.0/change-mgt-spec.html (visited on 05/2023).

[32] *OSLC Change Management Version 3.0. Part 2: Vocabulary*. May 2021. URL: https://docs.oasis-open-projects.org/oslc-op/cm/v3.0/os/change-mgt-vocab.html (visited on 07/2022).

[33] *OSLC Configuration Management Version 1.0. Part 1: Overview*. May 2022. URL: https://docs.oasis-open-projects.org/oslc-op/config/v1.0/oslc-config-mgt.html (visited on 07/2022).

[34] *OSLC Core Version 3.0. Part 1: Overview*. Aug. 2021. URL: https://docs.oasis-open-projects.org/oslc-op/core/v3.0/os/oslc-core.html (visited on 06/2022).

[35] *OSLC Core Version 3.0. Part 2: Discovery*. Aug. 2021. URL: https://docs.oasis-open-projects.org/oslc-op/core/v3.0/os/discovery.html (visited on 06/2022).

[36] *OSLC Core Version 3.0. Part 3: Resource Preview*. Aug. 2021. URL: https://docs.oasis-open-projects.org/oslc-op/core/v3.0/os/resource-preview.html (visited on 06/2022).

[37] *OSLC Core Version 3.0. Part 4: Delegated Dialogs*. Aug. 2021. URL: https://docs.oasis-open-projects.org/oslc-op/core/v3.0/os/dialogs.html (visited on 06/2022).

[38] *OSLC Core Version 3.0. Part 5: Attachments*. Aug. 2021. URL: https://docs.oasis-open-projects.org/oslc-op/core/v3.0/os/attachments.html (visited on 06/2022).

[39] *OSLC Core Version 3.0. Part 6: Resource Shape*. Aug. 2021. URL: https://docs.oasis-open-projects.org/oslc-op/core/v3.0/os/resource-shape.html (visited on 06/2022).

[40] *OSLC Open Project specifications*. URL: https://github.com/oslc-op/oslc-specs (visited on 07/2022).

[41] *OSLC Query Version 3.0*. Aug. 2021. URL: https://docs.oasis-open-projects.org/oslc-op/query/v3.0/oslc-query.html (visited on 07/2022).

[42] *OSLC Tracked Resource Set Version 3.0. Part 1: Specification*. Feb. 2022. URL: https://docs.oasis-open-projects.org/oslc-op/trs/v3.0/tracked-resource-set.html (visited on 07/2022).

[43] Jorge Pérez, Marcelo Arenas, and Claudio Gutierrez. “Semantics and complexity of SPARQL”. In: *ACM Transactions on Database Systems (TODS)* 34.3 (2009), pp. 1–45.

[44] *Schema.org*. URL: https://schema.org/ (visited on 08/2022).

[45] A Th Schreiber and Yves Raimond. “RDF 1.1 Primer”. In: (2014).

[46] Mohit Kumar Sharma. “A study of SDLC to develop well engineered software.” In: *International Journal of Advanced Research in Computer Science* 8.3 (2017).

[47] *SKOS Simple Knowledge Organization System*. URL: https://www.w3.org/2004/02/skos/ (visited on 08/2022).

[48] Steve Speicher, John Arwe, and Ashok Malhotra, eds. *Linked Data Platform*. URL: https://www.w3.org/TR/ldp/#bib-LINKED-DATA (visited on 08/2022).

[49] *The Semantic Web Made Easy*. URL: https://www.w3.org/RDF/Metalog/docs/sw-easy (visited on 08/2022).

[50] *VISDOM Data Management System*. URL: https://github.com/visdom-project/VISDOM-data-management-system (visited on 05/2023).

[51] Weiqing Zhang, Birger Møller-Pedersen, and Matthias Biehl. “A light-weight tool integration approach: From a tool integration model to oslc integration services”. In: *7th International Conference on Software Paradigm Trends, ICSOFT 2012; Rome; 24 July 2012 through 27 July 2012*. 2012, pp. 137–146.