An approach to automate the adaptor software generation for tool integration in Application/Product Lifecycle Management tool chains.

Student: Shikhar Singh
Examiner: Mihhail Matskin
Supervisor: Anne Håkansson
ABSTRACT

An emerging problem in organisations is that there exist a large number of tools storing data that communicate with each other too often, throughout the process of an application or product development. However, no means of communication without the intervention of a central entity (usually a server) or storing the schema at a central repository exist. Accessing data among tools and linking them is tough and resource intensive.

As part of the thesis, we develop a software (also referred to as ‘adaptor’ in the thesis), which, when implemented in the lifecycle management systems, integrates data seamlessly. This will eliminate the need of storing database schemas at a central repository and make the process of accessing data within tools less resource intensive. The adaptor acts as a wrapper to the tools and allows them to directly communicate with each other and exchange data. When using the developed adaptor for communicating data between various tools, the data in relational databases is first converted into RDF format and is then sent or received. Hence, RDF forms the crucial underlying concept on which the software will be based.

The Resource description framework (RDF) provides the functionality of data integration irrespective of underlying schemas by treating data as resource and representing it as URIs. The model of RDF is a data model that is used for exchange and communication of data on the Internet and can be used in solving other real world problems like tool integration and automation of communication in relational databases.

However, developing this adaptor for every tool requires understanding the individual schemas and structure of each of the tools’ database. This again requires a lot of effort for the developer of the adaptor. So, the main aim of the thesis will be to automate the development of such adaptors. With this automation, the need for anyone to manually assess the database and then develop the adaptor specific to the database is eliminated. Such adaptors and concepts can be used to implement similar solutions in other organisations faced with similar problems. In the end, the output of the thesis is an approach which automates the process of generating these adaptors.

Keywords: Open services for Life Cycle Collaboration (OSLC), OSLC adaptor, D2R server, Resource Description Framework (RDF), relational data mapping, OSLC4J meta-model.
Sammanfattning

Resource Description Framework (RDF) ger funktionaliteten av dataintegration, oberoende av underliggande scheman genom att behandla uppgifter som resurs och representerar det som URI. Modellen för Resource Description Framework är en datamodell som används för utbyte och kommunikation av uppgifter om Internet och kan användas för att lösa andra verkliga problem som integrationsverktyg och automatisering av kommunikation i relationsdatabaser.

Ett växande problem i organisationer är att det finns ett stort antal verktyg som lagrar data och som kommunicerar med varandra alltför ofta, under hela processen för ett program eller produktutveckling. Men inga kommunikationsmedel utan ingripande av en central enhet (oftast en server) finns. Åtkomst av data mellan verktyg och länkningar mellan dem är resurskrävande.


Keywords: Open services for Life Cycle Collaboration (OSLC), OSLC adaptor, D2R server, Resource Description Framework (RDF), relational data mapping, OSLC4J meta-model.
Table of Contents

ABSTRACT ......................................................................................................................... 2
Sammanfattning ................................................................................................................... 3
List of Figures ...................................................................................................................... 6
1. INTRODUCTION ........................................................................................................... 7
  1.1 Background ............................................................................................................... 7
  1.2 The Problem Statement .............................................................................................. 8
  1.3 Purpose ...................................................................................................................... 8
  1.4 Goal .......................................................................................................................... 8
  1.5 Methodology ............................................................................................................ 9
  1.6 Stakeholder .............................................................................................................. 9
  1.7 Ethics, sustainability and benefits ............................................................................ 9
  1.8 Delimitations ......................................................................................................... 10
  1.9 Disposition ............................................................................................................ 10
2. THEORETICAL BACKGROUND AND RELATED WORK ........................................... 11
  2.1 Resource Description Framework (RDF) ................................................................ 11
  2.2 RDF Schema ......................................................................................................... 13
  2.3 Tim Berner Lee’s 4 Principles ................................................................................. 15
  2.4 ALM/PLM toolchains ............................................................................................ 15
  2.5 OSLC Data standard ............................................................................................... 16
  2.6 Related Work ......................................................................................................... 19
    2.6.1 Previously developed adaptors ........................................................................ 21
3. IMPLEMENTATION ....................................................................................................... 26
  3.1 Working Method ..................................................................................................... 26
  3.2 Developing the Automation Approach .................................................................... 27
    3.2.1 Step 1: Mapping relational data to RDF .......................................................... 27
    3.2.2 Step 2: Creating the Adaptor model ................................................................. 35
    3.2.3 Step 3: Automatic Generation of the adaptor code using the Adaptor model .... 40
4. VALIDATION AND PERFORMANCE EVALUATION ................................................ 42
  4.1 Use Case: Sesamm Tool ......................................................................................... 42
  4.2 Adaptor Development ............................................................................................. 43
    4.2.1 Obtaining the mapping file. ............................................................................ 43
    4.2.2 Adaptor Model creation .................................................................................. 44
    4.2.3 Generating the Adaptor Code ......................................................................... 45
  4.3 Components of the developed adaptor .................................................................. 45
    4.3.1 JAX-RS ........................................................................................................ 46
List of Figures

Figure 1: Example RDF triple statements ................................................................. 12
Figure 2: Example triple statement with turtle format ........................................... 14
Figure 3: Example triple statement with multiple predicates ............................... 14
Figure 4: Example turtle statement with multiple objects ................................. 14
Figure 5: Tool integration using OSLC [26] .............................................................. 17
Figure 6: Concepts and specifications of OSLC core [28] ....................................... 19
Figure 7: Tool adaptor, as an interface to the tool .................................................. 20
Figure 8: Model based generation of Tool adaptors [36] .......................................... 21
Figure 9: Model based service discovery [35] .......................................................... 22
Figure 10: Tool-Adaptor structure .......................................................................... 23
Figure 11: Change management Specification [38] ............................................... 24
Figure 12: Communication between OSLC adaptor and Bugzilla ....................... 25
Figure 13: D2R server and the surrounding architecture. [53] ................................. 32
Figure 14: Structure of a D2RQ map created from a database table. [56] ................. 33
Figure 15: OSLC4J Meta-Model ............................................................................. 38
Figure 16: Eclipse provided GUI to model instances graphically ....................... 39
Figure 17: Generated Java classes corresponding to the meta-model ........................ 39
Figure 18: GUI for Sesamm Tool ........................................................................... 43
Figure 19: The mapping file (generated by D2R server) ........................................ 44
Figure 20: Adaptor Model(.XMI file) viewed via Sample Reflective Ecore Model Editor .......... 45
Figure 21: OSLC Adaptor Architecture ................................................................... 46
Figure 22: Code snippet showing OSLC annotations ........................................... 47
Figure 23: Data corresponding to table dboElementAnchorSet_UserFunctionAnchors in database. ... 48
Figure 24: Querying the adaptor for data using HTML browser .......................... 49
Figure 25: HTTP POST request via REST client to insert data in database ........... 49
Figure 26: JSON data sent as request .................................................................. 50
Figure 27: Snapshot of Sesamm database updated with the data via adaptor ......... 50
Figure 28: The layered architecture of the approach to automate adaptor development. [12] ....... 53
Chapter 1

1. INTRODUCTION

The aim of most of the organisations is to increase efficiency while at the same time reduce the time taken to manufacture an industry product. An underlying trend in this regard is to move towards automation of the most time consuming or redundant of processes, which include, but not limited to assembly lines or softwares. The thesis works to automate the development of an adaptor software. The thesis work is a part of the on-going research project ESPRESSO [66], which is part of a larger EU project ASSUME. [1] This research project is aimed at qualitative improvement and cost reduction of embedded system development in manufacturing environments.

1.1 Background

Product lifecycle management refers to the process of creating, preserving and storing the information that relates to the products and activities pertaining to the company so that stored data can be easily found, refined, distributed, integrated and re-utilized when required in performing daily operations in the company at a fast pace. [6] If the product being developed is a software, the process is termed as Application lifecycle management. A toolchain basically refers to a set of programming tools that work as a chain in the production of a software or a hardware/embedded product. Each of the individual tools in this toolchain focus on a particular stage of product or application development lifecycle.

In the linked data approach, introduced by Tim Berners Lee, one of the four defining rules states that one should serve useful information on the web for every URI so that classes and properties identified in the data can be looked up to get related information from generalized ontologies (sometimes referred to as vocabularies), including relationships between terms in the ontologies [11]. The approach is clearly applicable in solving our current industry problem. However, this approach mandates definitions and ontologies to link our data to, in order to provide them with concrete meaning and generalisation. OSLC may provide the missing link here. OSLC or Open Services for Lifecycle Collaboration is an open community with the aim of building practical standards and specifications that make interaction between tools possible. It specifies a common generalized vocabulary for different lifecycle stages like requirement management, test management, etc. along with other rules that all implementations must adhere to. [10]

During the development of a product in an organisation, it is important to have a holistic and unified view of all the data that is related to that particular product irrespective of the tool containing that data and its lifecycle phase. The data present inside these various tools managing the lifecycle of a product is difficult to integrate due to complex database schemas and varying user access. Varying user access is utmost importance here, as the organisation data is really sensitive and access to data needs to be strictly controlled. The user access and database schemas can still be taken care of when developing the desired adaptor for integrating data inside a particular tool with other tools by manually reading the schemas and user access rights. However, the integration becomes increasingly difficult when we try and automate the process of generating our desired integration adaptor/tool since these cannot be read manually anymore.
1.2 The Problem Statement

The thesis presents and investigates how can the generation of an OSLC adaptor be automated and how this automation process can be generalised, so that it can be used to generate an adaptor for any tool in an ALM/PLM toolchain. First challenge arising in this problem is how the data present in relational databases of various Application lifecycle management or product lifecycle management tools, prevalent in industries today, can be translated to semantic resource description framework (RDF) resources integrated with OSLC specifications and be presented as a graph of interconnected resources, so that this data can then be accessed by and linked to other tools and entities within these tools. Also, data inside the tools might be sensitive and private and hence, how data owner can control what data is given access to or is exposed, is also a question of research. The thesis then explains how the implementation of such an automation approach is realised technically.

1.3 Purpose

The purpose of the thesis is to help in communication and data exchange between various tools being used in Scania for development of their products by developing an adaptor. The thesis studies various approaches that can be a candidate to successfully map a relational schema to interconnected RDF resources with OSLC specifications and come up with the best possible technique. This technique can then be used in the development of a generalized approach that automates this process of data conversion and the OSLC adaptor generation.

It further aims to describe the development work of the architecture implementing the solution, taking a particular tool (SESAME tool in Scania) as a use case. The technical solution will also control the data being exposed from the schema, thereby solving the problems related to user access to some extent.

1.4 Goal

The penultimate aim here is to come up with a generalised approach that can be used for similar scenarios throughout the industry where integration between relational data sitting on various tools is required, and to validate the use of OASIS OSLC and linked data for such a solution. This will help towards the on-going research in the field of using linked data and OSLC for solving industry problems mainly of communicating data between various tools inside a toolchain. The approach taken for the prototype developed on a single use-case can be customized according to different integration scenarios to implement in any other environment.

The thesis will contribute to the Eclipse Lyo community, specifically towards the SQL4OSLC block. Whereby SQL data is persisted as a model and Java code is then generated from this model. [12]
1.5 Methodology

The research methodologies can be broadly classified into 2 categories namely the qualitative and the quantitative. Quantitative methodology approach is one of testing an already established theory via experiments or testing a system or algorithm by feeding in data sets and analysing the obtained result data set. Qualitative approach on the other hand is studying a phenomenon by doing exploratory research and background study to generate theories or solution prototypes. [7] For this particular degree project, the best suited approach is the qualitative approach since the methods that the thesis employs at various stages correspond to the ‘Qualitative research’ side of the research methods portal. [8] The thesis studies the current situation in a particular field and aims to establish an approach or algorithm which can then be tested by applying it on various use cases. Hence the research methodology is of the type qualitative here. A fundamental research [8] is carried out in the thesis where various applications are observed in order to obtain the insights on how the method of adaptor generation can be generalized. The thesis also hints at applied research since it is aimed at solving an existing practical problem in the industry. Based on the portal of research methods and methodologies described by Anne Hakansson [8], the thesis takes an inductive approach towards the research which is to be done. This is because the thesis involves observing already existing applications, opinions and explanations from industry experts in order to come up with the theories and the requirements for the product developed. For data collection, observations and interviews along with case studies are used. For quality assurance, measures like transferability, validity and replicability are discussed and employed. Replicability as the name suggests, refers to the ability of the research work to repeat itself when performed in a similar manner by other researchers. [32] Transferability means that the contributions of the research work can be used by other researchers. Validity conforms to the fact that the research has been carried in accordance with the existing rules. [32]

1.6 Stakeholder

The degree project is carried out at the Systems Development and Research (R&D) department of Scania CV AB, the Swedish automotive giant, which is headquartered at Södertälje, Sweden where around 3,500 people are employed. The company is mainly into the manufacturing of buses, coaches, trucks and engines. The organisation has around 45,000 employees stretching across 100 countries. [13]

1.7 Ethics, sustainability and benefits

The degree project at Scania has been approved and allowed, subject to Scania confidentiality conditions, whereby any software material developed during the degree project is property of Scania AB and has not been published in this literature. Also, any inventions achieved will be considered A-inventions according to agreements on the rights to employee inventions entered into by the Swedish Employer Association (SAF). The data and business information specific to Scania is also not shared in this literature due to the non-disclosure agreement.
Since the project aims at automating the adaptor generation for various tools in a business environment, so that the task can be performed via softwares which otherwise requires human effort will go a long way as it cuts the human involvement in the process, thereby reducing both time and human resource costs for the organisation. It also makes the process more efficient and exposing data more controlled. Moreover, since technicians no longer have to manually look at the tool data to generate the adaptor, chances of data confidentiality breach due to human error also decrease. The performed work as a part of the bigger ‘ASSUME’ project will further the research on RDB (relational database) to RDF conversion.

Since the degree project is being undertaken in Scania which is using these toolchains in the manufacturing of trucks and buses, an efficient toolchain will directly contribute to the production of safer driving vehicles, in turn resulting in decreased accidents and loss of life and property.

### 1.8 Delimitations

The effort to develop the approach if successful will considerably improve the integration of various tools in a product or software toolchain. The time and cost expended on the life cycle of a product will come down drastically. In the context of the organisation where thesis is implemented, improvement in the production process of vehicles and embedded systems directly converts to a more efficient and safer product. The thesis however is not a comprehensive work for implementing the adaptor. It automates certain crucial parts of this adaptor only. Also, the thesis work can be replicated only for similar scenarios that exist and does not solve automation problems in all the industry scenarios. The approach developed in the thesis works only for tools with SQL databases. Major changes might be needed for it to work on tools with different databases.

### 1.9 Disposition

The complete thesis has been divided into 5 major chapters. The current chapter sheds light on what is the research focus of this thesis, the description of the existing problem, aim of the study, the methodology used to realise the goals and the sustainability aspects of the work. The next chapter discusses all the important concepts needed to be understood in order to have a good hold of how the thesis work is conceived and how the application is developed later on. It also discusses the related work performed in the field so far. The third chapter discusses in detail, the different steps of the developed approach that successfully automate the adaptor generation. The subsequent chapter, which is the 4th chapter, describes the validation of the developed approach by implementing it on a use case and making an adaptor, followed by a performance evaluation of the developed adaptor. In the final chapter, the work is concluded and future work along with its scope is discussed.
Chapter 2

2. THEORETICAL BACKGROUND AND RELATED WORK

The problem at hand, of developing the approach to automate the generation of OSLC compliant adaptor for various tools in an ALM/PLM toolchain, pivots around the question of translating the relational data present in the databases of these tools into semantic linked resources. This the initial question that needs to be answered in the quest for automating the adaptor. The goal of representing stored relational data as connected resources can be achieved via a number of approaches. All however involve an important concept known as resource description framework (RDF), which is a framework or set of guidelines to describe information about resources that will be generated from the relational data and then linked together. Linked data is another concept that is important to understand here. It is the coming together of huge amount of data present in a specific format that are related to each other with semantic web technologies like RDF, OWL, etc. and possibly understood by semantic tools. RDF and some other related concepts important to attain the goal are discussed further.

2.1 Resource Description Framework (RDF)

Resource description framework, also referred to as RDF, defines a set of rules to express any piece of information that needs to be exchanged. The standard is crucial when the data exchange takes place between applications and needs to be in a machine readable format while making sure that no meaning is lost. In the language of RDF, every abstract concept or document or object is referred to as a resource.

RDF finds its applications in a myriad of applications. The most prevalent use is to add machine readable information to already existing web pages on the internet. Building of social networks which are distributed in nature also use RDF framework by linking information about individuals across multiple distributed social networks. RDF also help in the development of semantic web whereby the data stored in various formats can be accessed by web applications as RDF data.[14] Now RDF framework specifies an RDF data model to store data. In this data model, all the information about the resources is stored in the form of a triple. A triple is a statement consisting of 3 parts just like any other statement in the English language. It consists of a subject, followed by a predicate which is then succeeded by an object part. [18] The ‘resource’ in the RDF can be placed in the triple at the object or subject position. Since a resource can be placed at both subject and object positions, a web of interconnected resources can be discovered. This happens for example when a resource is an object in one triple while the same resource is subject in another triple. One can hence browse logically from one resource to another giving rise to an interconnected graph where resources form the nodes of the graph. The predicate is usually referred to as a property which specifies the relationship between the object and the subject. Thereby forming the edges of such a discovered graph. [18]

The graph also acts as a tool to bring together information from different sources in a useful way. Thereby interlinking numerous datasets online. However care needs to be taken while
bringing together this information because this process of fusion might induce some incoherencies in the data which is being worked upon. Hence we need to stick to some basic guidelines. One of them being that, each triple is termed as being 'true' if the relation portrayed between the subject and the object actually exists. And correspondingly the RDF graph will be termed as 'true' if all the containing triples in it are true. [22] The concept of an RDF triple statement being true gives rise to another powerful concept. The power of making logical deductions from already given statements. Given a certain set of statements, a system can deduce new statements which will be logically true. Not only can new statements be deduced, but one can also find if all the given triple statements are in coherence with each other or if they are in contradiction with one another. An example of such a set of statements is given in figure 1. Here statements A and B are independent of each other and true in nature. And by assessing statements A and B, it is logical to conclude statement C.

**Statement B:** <http://example.org/#Yan><http://purl.org/vocab/relationship/childOf><http://example.org/#Walsh>

**Figure 1: Example RDF triple statements**

The discovery of such a graph is possible by querying the data stored in datasets in RDF format using SPARQL query language. The three parts of a triple can be in the form of an IRI. IRI stands for International resource Identifier and can be used to describe both the resource or the property connecting them. URI is a type of IRI. IRI may not mean anything alone, but when combined with a vocabulary or convention, they get a defined meaning and can be used to represent a resource. IRIs are reusable and hence they can be used again and again for representing the same thing at different locations. This makes the scope of an IRI global when used in a triple. [22] There are two more things that can be found in a triple. One of them is a literal which is a solid value used only at the object position. A date like ‘23.09.2016’ or a string like ‘Barak Obama’ are all examples of a literal. The other one is a blank node. It is a resource which is present in a triple but is not related to a vocabulary and is denoted without a global identifier like IRI. It has a local scope.

Another integral part of RDF data model which is specified under RDF is RDF vocabularies. These vocabularies help give meaning to the resources defined in the RDF probably with the help of IRIs. They are used for defining different terms and categorising them and also to provide constraint information about these terms. Most of the times, the terms ‘Vocabularies’ and ‘ontologies’ are used interchangeably and as such there exists no specific differentiation between the two. These vocabularies contain more information on the defined resources and provide a semantic meaning to these resources and form the basis for inference of semantic data. Vocabularies help resolve confusion when same term appears in different data sets and the definition of terms in the vocabularies leads to discovery of new knowledge via inference. An example of such a vocabulary is Friend of a Friend (FOAF). Foaf is a dictionary containing definitions of a lot of people related terms that can be used in RDF world. It defined many named properties and classes by utilizing the provisions of the W3C’s RDF. [15] Most of these classes and properties have an associated IRI, which can be used when defining a resource of this class or property type. Since it defines a lot of properties and classes related to people, it finds broad usage in social networking websites along with anything else that needs to maintain information about people and link that information together in a logical manner. According to the official description of FOAF, ‘It integrates 3 kinds of networks, a network of social interactions between humans, their friendship and relations; and networks representing the information about this world that is generated.
FOAF again describes a number of terms that can be categorised as either a class or as a property. Most of the FOAF vocabulary terms are defined using computer language like RDF/OWL so that they are easily processed by softwares. The unique quality of FOAF is that many FOAF documents can be combined together in order to generate a unified information database.

Other examples of vocabularies are **SKOS (Simple Knowledge Organization System)**, which is used for publishing classification schemas like the Thesauri on the web. Due to the classes and properties it defines, it has been widely accepted in the library world. In 2009, it became a W3C recommendation. [16] Some other prominent vocabularies used widely are **Dublin Core, schema.org, etc.** Another prominent vocabulary in use is the OWL2 vocabulary. The exchange syntax for OWL2 is RDF/XML.[23] OWL2 consists of a number of sub-languages that offer numerous computation and application benefits. OWL2 EL is the one that helps with applications employing really large ontologies by supporting polynomial time algorithms for reasoning problems. OWL2 RL is the most important of all that supports polynomial time reasoning algorithm using numerous database technologies. It operates directly on RDF triples and is specially useful where light-weight ontologies are employed.[23] This vocabulary is unique in the sense that it provides certain terms that help bring different vocabularies together. For example, sameAs property defined in the OWL vocabulary can be used to specify that two terms defined in 2 different vocabularies are actually pointing to a same thing/resource.

### 2.2 RDF Schema

Definition of vocabularies such as FOAF, SKOS, etc. is done according to some rules which are specified as a part of what is known as RDF Schema or schema language. With these rules one can specify guidelines pertaining to how RDF vocabularies should be defined and how they should be used. One example of such a rule is the construct of class to define various categories that are used for classification of resources. According to the RDF schema document given by W3C, some of the other main constructs provided are **Property, Domain, Range, Type, etc.** [17] **Property** is something that specifies the relationship between 2 resources and how they are connected. **Type** is a property that specifies relationship between an instance and its class. **Domain and Range** help put type restrictions on what a subject or object in a triple can be. The class and property constructs defined by the RDF Schema are similar to the class-property concept present in any other object oriented programming language. There exists however a fundamental difference in the way this concept is applied in RDF. While in object oriented programming the class is defined in terms of the properties its instance may have, in RDF schema it is the other way round. A property is described as the classes to which it might relate to. [17] A clear advantage of such an approach is that more properties can be defined for a particular class without requiring editing the definition of the class again. Also, the fact that anyone is allowed to edit or enhance the description of the resources, puts the RDF approach in coherence with the principles of the web defined by Tim Berner Lee. [19]

RDF schema also provides with a lot of RDF syntaxes such as TURTLE, TriG, JSON-LD, etc. TURTLE for example, is another RDF Triple Language which is based on the RDF syntax. A TURTLE document is a textual representation of an RDF graph. An RDF graph in its structure is triples consisting of subject, predicate along with an object. In the TURTLE language, each of the triple is terminated by a `.`. An example of a triple written in TURTLE
format is described in figure 2. [21]

Figure 2: Example triple statement with turtle format.

It could be the case that a single subject is referenced by a number of predicates. In such a case, a turtle representation lists the subject only once followed by all the predicate-object pairs, each separated with a semicolon (;). An example of such a representation is figure 3 [21].

Figure 3: Example turtle statement with multiple predicates.

It could also be the case that a subject-predicate pair is often referenced by a number of objects. In such a situation, the subject-predicate pair is written down followed by all the referencing objects which are separated with a comma (,). Figure 4 lists an example of such a case. [21]

Figure 4: Example turtle statement with multiple objects.

The TURTLE format also offers a number of ways in which RDF related elements such as literal or an IRI can be written. A TURTLE document is said to be conforming if it is a Unicode string that follows the rules and grammar constraints defined in the TURTLE grammar written in the W3C TURTLE specification document. This TURTLE document can then be used to serialize RDF graph. This TURTLE document can then be read by any system or application using what is known as a TURTLE parser that provides the serialized RDF dataset to the application. A turtle can also be embedded in an HTML document using the script tags by assigning the value ‘text/turtle’ to type attribute.

A concept important to know in relation to RDF is Reification. Reification is the process of making RDF statements about another RDF triple statement. [20] This functionality comes into play when an application wants to record details about another RDF statement. This in some sense is just like storing metadata about the RDF statement like details about the creator or the source of that information. An example of such a situation is when there exist an RDF statement and the application wants to make a statement about where that information came from i.e. who wrote that statement. [20] To undertake the task of reification, RDF provides with a vocabulary meant to describe RDF statements. The RDF vocabulary for reification has the properties rdf:subject, rdf:predicate and rdf:object. And also consists of the type rdf:Statement. These together form the reification quad and are necessary whenever a statement about another RDF statement needs to be made. Then other RDF statements could be added to this quad in order to associate more information with the RDF statement being described. An important distinction here is that asserting the reification does not mean that we are asserting the original statement being described. Another important thing to keep in mind when using this reification vocabulary is that it is easy to construe that the vocabulary is defining something that is not actually defined. This can be avoided by following some more guidelines on how to use RDF reification. Reification explains what is the relationship that exists between the resource a triple might refer to and a specific instance of the triple.
2.3 Tim Berner Lee’s 4 Principles

The concept of linked data is one where information placed on the web is linked to each other, so that when one has some information, he or she can browse to other related pieces of information on the web. This information exists on the web in the form of documents and is linked together using RDF. Now in the world of web, URIs can be used to signify anything from an abstract concept to an actual person. This power of URI can be combined with RDF concepts to link data. Tim Berner Lee has given 4 guidelines that help link data on the web by using the RDF and URI as the supporting pillars. [24] The first being that, URIs should be used for naming things. Then, these URIs should be HTTP so that anyone can look them up. However, this guideline is sometimes undermined due to the invention of new URI schemes. Now, if someone looks them up, provide more useful information using the linked data standards such as RDF. [24] This points at providing more information about the classes and properties that are being utilized. The last important rule suggests that apart from providing useful information, links to other related URIs shall be given so that new things can be discovered. [24] This rule helps in bringing to life the idea of the web of information similar to the web of hypertext documents.

Using these guidelines is not mandatory, but increases the usability of data on the web drastically and in ways unimaginable. The linked data forms a web where multiple RDF statements link multiple resources. These resources act as the nodes in the graph. So when someone tries to look up a URI corresponding to a resource, the server will return all the details of the particular node. This information comprises of 2 things: all the blank nodes that are associated with this resource and all the other resources which are associated with this resource. This can be found out by looking at all the RDF statements where the node corresponding to the referenced URI is either a subject or an object. Hence it is mandatory that an RDF statement that links together two documents be present in both these individual documents. This way both the documents are reachable starting either of the two.[24] This however gives rise to the problem of inconsistency and we cannot just simply duplicate the RDF statements. Hence different techniques are employed to keep data consistent yet browsable. We can ignore some types of links due to their sheer number or we can put links of a certain type in a completely new document and specifying it with another statement.

Linked data can be classified as open or closed. Open linked data stands for linked data that is released with an open licence and is free to be reused by anyone else. Linked data can also be used internally in an organisation or for personal use. This falls into the category of closed linked data. More recently a scheme of assigning stars to linked data has been in place. This scheme rates the linked data on its openness and its ease of usability. [25] The first star is achieved merely by giving an open licence for your linked data. If the data is in a machine readable format, then a second star is assigned. The next one is achieved when the data format being used are also open source. Using more open standards specified by W3C for example RDF wins another star. The fifth and final star is assigned when your linked data is also connected to other linked data on the web so as to give some context to it. [25]

2.4 ALM/PLM toolchains

Product lifecycle management (PLM) refers to the whole process of managing a product right from the time it is an idea to the final delivery of the finished product including succeeding activities like service, etc. Now when the product being developed is a software or a computer application, that involves release management, computer programming, etc. the process is termed as application lifecycle management. An application lifecycle may contain many phases namely requirement gathering, product planning, development phase that
further contains designing the code, coding, code reviewing, unit testing and integration testing, which is followed by build management, application performance management, end-user experience monitoring, maintenance management and so forth. During the process of software development, a developer uses number of tools that support the developer during various phases of development like designing, code generation, testing, etc. The chain formed by the integration of tools relating to the further phases of the process is what is known as a toolchain. [30] A toolchain is also referred to as a software that tries to integrate the different tools involved in the development of a product. [31] An important task when talking about these tools is how to integrate these tools and give rise to these toolchains. Now since tools that might form a part of the toolchain are distributed across different platforms, the toolchain that is developed also becomes a distributed system. Hence, the integration involves communication and synchronization between heterogeneous platforms. Integrating a toolchain involves numerous aspects. One of them is integrating the data stored inside these tools and also controlling how data within different tools relate to each other. Integrating the control process i.e. how different tools notify and activate each other. Process integration refers to providing data to process management tools from other tools like development tools. One of the most important aspect is platform. It takes care of providing a virtual environment for different tools which work on different hardware and software, to operate. An approach to overcome such an obstacle effectively is to develop adaptors for each individual tool in the toolchain. These tool adaptors effectively expose the functionality provided by the tools as a service to others. In this way the tools’ functionality can be used by other tools irrespective of the platform and other heterodox systems. Hence these adaptors can be viewed as a wrapper for each of the tool present inside a toolchain. [31] Now for the development of such adaptors (also referred to as wrappers) for tools present inside a toolchain, some ground rules need to be laid since the platforms, concepts and the technologies involved in these tools differ widely. These rules involve using standards and concepts defined in this regard, such as OSLC, RESTful services, etc. Some of these concepts are discussed in the succeeding sections. These ground rules help develop adaptors for different tools regardless of their platform or the technology they are using.

Now, attempts have been made previously to integrate a tool-chain via various ways possible. Some of the approaches imbibed the use of a well-defined process. But since this process used a lot of non-standard technologies at various steps, it was not suitable for a generalized application. [30] However, after recurring application of such different approaches, certain patterns corresponding to the process of integration emerged. Some of these patterns can be classified into what is known as the meta-model based tool integration approach. The idea behind this approach is to maintain a data repository which consists of all the meta information about the data that is exchanged between the tools in the toolchain. Once this information is readily available it can then be used for supporting the data exchange and other activities between the various tools inside the toolchain. An existing application of the meta-model based integration approach is discussed in the related work section later on.

### 2.5 OSLC Data standard

OSLC, which stands for Open Services for Lifecycle Collaboration, is an open source community that actively builds specifications in order to integrate data flow between softwares being used in the industry. It focusses on softwares being used in the software development lifecycle and helps integrate data flow between them so that this seamless data flow can help speed up and make efficient the software development process on the whole. This automatically converts into increased profits for the company. The specifications that
the community comes up with, provides conforming lifecycle tools to integrate workflow of development lifecycle processes. OSLC forms different workgroups that focus on particular domains of software development and operations for example change management or embedded systems and comprise of industry experts from that domain. [26] The task of each work group is to focus on a particular domain and work on issues pertaining to that domain explicitly. One of the major tasks involves defining vocabulary for terms and concepts used in that particular phase of the lifecycle that in turn helps in integration of tools in the product lifecycle. All the specifications developed by the organisation are based on only standards prevalent in the industry. For example the services developed by the working groups are RESTful and resources being defined can be accessed by a unique URL. Since it is an open source community, all the specifications and tools developed by the working groups are free to use and easily integrate with the tools already in use.

For this integration of data between tools, OSLC uses semantic data and RDF as the underlying concept.[26] In OSLC, every entity is HTTP resource identified by a URI. Figure 5 described how data between tools is integrated using OSLC and linked data.[26]

Usage of linked data in data integration increases the analysing capability and exploration of new data. Apart from the different work groups that focus on the individual domains (phases of development lifecycle), there also exists a core workgroup. This group is bestowed with the responsibility of defining core specifications. The OSLC core specifications consist of rules that guide how the specifications for each individual domain be developed by the domain groups. Usage of these rules help in better integration of lifecycle tools. The core specifications majorly list how domain groups should use HTTP and RDF. The core specifications hold no meaning alone. The core specification when combined with the domain specification give rise to the OSLC protocols that are utilized by the domain tools. OSLC is devoted to creating specifications via which tools can interact with each other. For interaction via OSLC protocols, software tools need to follow one or more rules defined in the OSLC specifications. It is not mandatory that all the rules and guidelines specified in the specifications be followed in order to attain this inter-tool communication. [27]

Now integration of data takes place via two primary methods which are offered by OSLC. In one of the techniques, data is linked via embedding HTTP link to one resource in the representation of the other resources. In addition to that, OSLC defines protocols for
communication between the tools that implement this common OSLC specification. This communication comprises of retrieving the data stored in another tool, exposing the data stored in itself according to the queries sent by other tools or deleting and updating data according to the requests coming in the form of HTTP. This method however focuses on tool to tool communication and is not user friendly since the operations are performed based on the HTTP requests coming from other tools are not human interpretable.

In order to make it more user friendly, OSLC also provides for data integration by providing HTML user interface. OSLC defines protocols that enable a tool to display data stored in it. This web interface enables humans to understand data inside a tool and make links to this data thereby making the integration process smoother for humans.

Figure 6 describes what relationships and concepts make up the OSLC core specification. As explained earlier, the OSLC core specifications describe various features that could be present inside an OSLC service formed by utilizing the core and domain specification. It also describes the behaviour expected of an OSLC client. A term that needs to be defined here is a resource. A resource is a network data object or a service that can be identified by a URI and might be represented in a lot of different ways. [28] The OSLC serviceprovider resource lists all the OSLC services that are being provided. All the service providers are in-turn being listed in what is known as a serviceprovidercatalog. Each service provides with three basic functionalities to operate on a resource. A creationfactory that supports creation of a resource and a querycapability that provides with the functionality to make requests to the tools to query resources. A third functionality known as a delegatedUIdialog provides to the users, the capability of creating a resource and linking more resources to an already existing resource by using a web interface. Another entity known as resourceshapes exists inside the creationfactory and the querycapability. The properties of the resources managed by the service are written inside it. Another important role of OSLC is to set out the guidelines on how the resources defined in OSLC will be represented in various formats like RDF OR Turtle. [28] OSLC also utilizes the Extensible Markup Language(XML) namespace mechanism in the definition of different resources. OSLC must also follow the HTTP specification for operations on the resources. OSLC uses these HTTP specifications for performing the create, retrieve, update and delete operation on the resources inside the plethora of tools following the OSLC specifications. [28]

As far as authentication protocols are concerned, no access control or authentication approach is required for usage of OSLC. However, it recognizes OAuth protocol and provides for its usage by defining a property in both ServiceProvider and ServiceProviderCatalog that might be used to hold configuration values for OAuth and by defining a resource called OAuthConfiguration to hold the 3 URL values needed for token negotiation in OAuth. [29] OSLC nowadays is widely being used in multiple softwares being developed by a lot of big organisations. This includes companies like IBM, oracle, Alcatel-Lucent and NASA jet propulsion Laboratories, just to name a few.
2.6 Related Work

Internet in its nascent stages was considered to be a web of documents. These documents were nothing but HTML pages and websites. These documents were interconnected with what is known as hyperlinks. These hyperlinks connect 2 documents and help navigate the user from one web document to another. A hyperlink is the address of a web document, called as head, embedded in another web document, also referred to as tail. [2]

However, Internet has now changed drastically from just being the 'Web of documents' to being 'Web of data'. This has been made possible by exposing the data stored in databases as entities on the internet. The process makes use of concepts such as Resource description framework(RDF), Uniform Resource Identifier(URI), HTTP, etc. Each data is represented as a resource and data related to each other are portrayed as linked resources. A data entity, shown as a resource, is also connected to data that gives the meaning or definition for this data entity. Thereby making the internet Semantic Web i.e. internet where almost everything has a meaning.

Semantic web has found applications in a plethora of fields. Concept of Semantic web combined with the power of machine processable information, improves the mining of web itself by exploiting its semantic structure. [3] Another application is in designing of web portals semantically, which act as tools for information presentation and exchange over the internet.[4]

The concept of semantic web is one that has unseen applications outside the world of internet as well. One such application is the creation of semantic sensor web. Sensors are distributed across the globe, producing tons of data about the environment. However, the data generated by these sensors cannot be used to deduce useful knowledge owing to incoherent communication. As a solution, the sensor data can be annotated with semantic metadata that will help provide contextual information for situational knowledge. [5] The
same concept can be applied to the databases of various tools that form the toolchain of an ALM/PLM.

In almost every organisation, a product or software lifecycle is managed by a number of tools working in line with each other forming a chain. The data exchange between these tools is not well integrated. In order to extract data from another tool, each tool is dependent on a central entity that is aware of the schema of all other individual tool databases. Or the tool wanting to extract data needs to fire a query on its own to get hold of the required data. This however is time and resource intensive.

A way to integrate the tools in the tool chains is to describe the toolchains as models on a higher abstraction level. [35] We then use this model in combination with model-based techniques to develop adaptors that act like wrappers to the tools we want to integrate in the toolchain. The integration of tools in a toolchain involves majorly 2 aspects. Sharing of data inside the tools with other tools in the toolchain and exposing the functionality of the tool for other tools. This wrapper (tool-adaptor) helps provide access to both data inside the tool database and operations that the tool offers, as web services. Figure 7 depicts how a tool adaptor acts as a wrapper to the tool and works as an interface for the tool to the whole toolchain.

![Figure 7: Tool adaptor, as an interface to the tool.](image)

Now, one of the approaches for the development of such tool adaptors that expose tool data and its functionality using model based techniques is specified in the work of Martin Törngren et al [36]. The adaptor development consists of an EMF Tool model development. As shown in figure 8[36], OSLC resource shapes and OSLC service definitions form other parts of the tool-adaptor that needs to be developed. All these elements use the EMF meta-model obtained directly from the tool. Further discussion and explanation of what an EMF is, is done in the next chapter. This adaptor then, as explained earlier, acts as an interface for the tool to communicate with other tools in the toolchain. This adaptor (adaptor 1) communicates with adaptors of the other tools (like adaptor 2 an adaptor 3) in the toolchain, as shown in the figure 8.
Figure 8: Model based generation of Tool adaptors [36]

The wrappers for every tool can communicate with each other since they are developed on the same underlying concepts and technologies, thus integrating the toolchain. Two major parts of integrating the tools using the developed tool-adaptors via the OSLC approach are the service discovery i.e. discovering the services offered by other remote tools that need to be integrated and the orchestration[35] of these services i.e. using these services to our ends. Figure 9, as explained by Matthias et al. in their publication [35] describes the approach to service discovery. The discovery is aimed at finding out all the details about the remotely deployed adaptors. This is done by parsing the metadata that is obtained by following consequent links, starting from the entry point URL. This is the URL for the `serviceprovidercatalog` as discussed in the previous sections. From this URL we obtain the URL for `ServiceProviders` and `ResourceShapes` which are parsed as well. The meta-model is extracted using all these 3 major RDF resources. The meta model extracted acts as a description of what data is contained inside the tool and what services are offered by it. Another use of the metamodel obtained for the tool adaptor is that it could be used to verify the service description that is discovered with the model being used in the orchestration model. This check becomes crucial when we want to develop code based on this metamodel. It helps root out inconsistencies at the model level which are comparatively easier to detect when compared to detection at the code level.

2.6.1 Previously developed adaptors

Now, many adaptors have already been generated and tested using the model based approach to develop adaptors and integrate the tools in a toolchain. Generation of tool adaptors help in bringing together all the tools in a toolchain. It also helps add a new tool
and expand the already existing toolchains. A tool-adaptor for a tool X, enables other tools in the toolchain to ADD and DELETE resources or QUERY resources inside tool X. The adaptor should be able to handle different types of request namely, RDF/XML, HTML, JSON, etc. and should produce responses of appropriate type as well. However, developing these adaptors is a cumbersome task since an agreement needs to be reached between the tools and we also need to know the properties of relationship between these tools. Also, we need to have a large amount of information about the tool for which the adaptor is being generated.

2.6.1.1 The MATLAB/Simulink adaptor

In the work of Martin Törngren et al. [36], an approach to automate the tool adaptors is discussed. They also develop an adaptor for MATLAB/Simulink tool based on this meta-model approach. MATLAB is a tool that is used in the development of embedded products. It provides the ability to model the control functions and their simulation. Now in the development of the said adaptor, the first step is a specification mechanism. This specification consists of a lot of details about the tool. An important part of which is the tool data. Now for specifying the details of the data, the authors use meta-data modelling technology of EMF. In this, the data is described inside the ‘EPackage’ as a directed graph. The nodes in this graph are of type ‘EClass’ and ‘EAttribute’ and the links between the nodes are ‘EReferences’. [36] This graph is to be developed manually by reading the data stored inside the tool’s database. This is one of the solutions developed as a part of the thesis, where there is no need to read this data manually from the database for the specification (part of which is the meta-model). The specification developed by the authors however is independent of the technology for implementing the tool adaptor. The authors automate the generation of the code for the tool adaptor from the specification onwards. The authors classifies the generation of the code in two parts. One of these parts is the generation of the code for integrating the adaptor in the toolchain. The second classification is the code that
interacts with the tool. Figure 10 specifies the two parts of the tool adaptor formed by these 2 kinds of codes.

![Figure 10: Tool-Adaptor structure](image)

'TA internal' refers to the code part that interacts with the tool. This interaction can take place via an API offered by the tool or can take place directly. An example of the adaptor communicating with the tool via an API is the Bugzilla adaptor that is developed for the bug tracking tool Bugzilla. The internal part of this adaptor communicates with the Bugzilla database via an API which is provided by the developers of Bugzilla. More of the Bugzilla adaptor is discussed in the later sections. 'TA-External' refers to the code part that interacts with the other tools and the toolchain and helps integrate this particular tool into the toolchain. Now authors refer to the generation of the code corresponding as semi-automated generation. In this semi-automation of the tool generation, there are many parts that need to be performed manually. One of them, already discussed earlier, is the generation of the meta-model from the tool. Another major part of the manual work to be done is to add the implementation code to the code skeleton developed by the authors.

The authors then move on to the discussion of fully automated code generation of the adaptor. This technique however can only be applied to tools that are EMF-based eclipse tools.[36] The authors develop an approach to generate the tool-adaptor automatically that comprises of 4 steps. The first step is to define an EMF based tool meta-model. This step is a manual step where the EMF meta-model is defined by the user wanting to develop the tool adaptor. The whole approach depends on this step as mentioned by the authors themselves. This step provides the input to the whole adaptor development process [36]. This thesis automates this crucial and lengthy step of generating the EMF meta-model from the tool for the generation of the tool adaptor. It saves the developer's effort and time spent to read the tool database and its structure to generate the meta-model. The second step is to use EMF along with the tool meta-model to generate OSLC resource shapes, ServiceProvider and ServiceProviderCatalog. The third step involves using all the generated elements like the ServiceProvider and the ResourceShapes to generate code skeleton. This is achieved using EMF and related model to text conversion technologies. The fourth step is of putting in the implementation code inside these skeleton classes. This is to be done manually as well as carefully since it depends on what functionality is needed and what technologies are to be used. The thesis however automates this step of writing the implementation code inside the skeleton classes as well. Thus further bringing down the process of automation many folds. How this is done is discussed in the upcoming chapters.

Hence, these steps are adopted in the approach developed in the thesis for the generation of an adaptor with considerable changes, in order to make the process more automated, fluent and less effort intensive.
2.6.1.2 The Bugzilla adaptor

Another adaptor that has been developed on similar concepts of meta-data, etc. is the one developed for bug tracking tool Bugzilla. Now Bugzilla is a widely used tool inside organisations for tracking defect and code changes, submit and review patches, etc. Bugzilla is an open source tool and costs nothing and provides various features that other paid tools provide.[37] By developing this adaptor for Bugzilla, it is added into the Linked data Platform(LDP) where it might work in collaboration with other tools with OSLC to give rise to an integrated toolchain. Due to the functionality and features that the Bugzilla tool offers, OSLC change management specification [38] is needed to give Bugzilla OSLC support. It is important to build this adaptor for Bugzilla since it does not have any built in OSLC support unlike some other tools that do have OSLC support provided in them like Rational Team Concert (RTC).[40].

Change management specification defines a RESTful web services interface for change management, managing the change requests, activities, tasks of various products and relating these with other resources such as project, category, plan, etc. [38] The specification however is only meant to define some capabilities that may be used in integration scenarios defined by the change management working group and does not give a complete detailed interface to change management. [38] Change management specification uses the OSLC core concepts along with referencing resources defined in other domain specification.[38] Figure 11 illustrates this. Now, any tool can be converted into either an OSLC consumer or OSLC provider by developing the adaptor. A consumer is the part of the adaptor that acts as a client. It consumes the OSLC service provider services so that it can access the domain data via delegated interfaces and service calls. A service provider is the part of the adaptor that acts as a server. It exposes the data inside the tool in accordance with the OSLC specification. Consumer part of one adaptor uses services provided by the service provider part of another adaptor. The consumer requests deletion, update or viewing of data from the service provider.

![Figure 11: Change management Specification](image)

Now, a Bugzilla adaptor is a RESTful web application where all services are handled by a single JAX-RS servlet. It is build using OSLC4J. This is a Java toolkit that is used for the development of OSLC providers and consumers. OSLC4J consists of a core component, that provides with OSLC annotations and model support, along with a Apache Jena and Apache Wink provider.[39] Every change management specification resource is defined as java class. Apart from this, the adaptor also consists of code that connects to the Bugzilla server using J2Bugzilla API provided by the developers of Bugzilla. This acts as a way to get hold of data inside the Bugzilla database and other features offered by Bugzilla. As shown in figure 12, the OSLC support is rendered to the Bugzilla server using the adaptor that acts as an interface for it. The OSLC request that come in from other tools(in the form of HTTP PUT,GET,POST requests) are entertained by the adaptor that consists of java code based on OSLC specifications that in turn uses the Java API to connect to Bugzilla and access its
elements and then displays the items using *Delegated Uls*.[37] However, to develop the Bugzilla adaptor, just like the MATLAB/Simulink adaptor, the metadata of the Bugzilla tool needs to be read manually by the developer and then mapped with the OSLC specifications. Also, the generation of OSLC adaptor code can be automated up to only a certain level. We can create the skeleton classes for the code. But the implementation inside the code needs to be written manually since this implementation needs to call and make use of the functions defined in the Bugzilla4J Java API. Hence, we can conclude that there exists a definite technological gap when it comes to automating the generation of an OSLC adaptor. This technological gap is addressed in the work carried out as a part of this thesis. The steps to generate the OSLC adaptor can be automated completely with the solution offered in the thesis.

2.6.1.3 IBM Rational - JIRA Adaptor

Jira is a very diverse tool, developed by Atlassian, that lets teams perform a variety of tasks. It is used in tracking of bugs and tasks, it relates different issues to the source code, report the status of the project and most of all plan agile development process. It also helps you monitor build statuses. JIRA can also integrate with already existing tools like Confluence, Hipchat, etc. It also helps you design your own workflow for a project or chose from a variety, already provided in JIRA. [41] Due to the nature of tasks it performs, an OSLC adaptor developed for JIRA must implement Change Management, Quality Management and requirements management specifications. [42]

IBM has developed an OSLC adaptor for JIRA software. This adaptor helps integrate JIRA with the IBM-Rational lifecycle tools like IBM Rational requirements composer, IBM Rational software architect design manager, etc. [42]. Neither the tools are open source and free nor the adaptor developed by IBM. The adaptor that is developed by IBM uses Project Lyo which is an open source project that provides with an SDK to develop OSLC enabled tools. This is discussed further in the upcoming chapters since this technology is used in developing the automation approach of the thesis as well. The adaptor developed by IBM also uses Jazz as an integration platform. The approach is to create a link between a JIRA issue and an RTC Work Item or between JIRA issue and RRC requirement. For this change, quality and requirement management specifications are used, as described earlier. The adaptor developed, enables JIRA as a service provider wherein Rational CLM tools can create and access Change Management data in Atlassian JIRA. The approach developed by IBM also enables to integrate any other 3rd party tools into the lifecycle toolchain for example HPQC.
Chapter 3

3. IMPLEMENTATION

3.1 Working Method

1. The first step is realised by thorough literature study of the existing information about the domain to attain comprehensive background knowledge and chalk out specific aspects of the problem statement that will be worked upon. The focus then shifts to constructing a theoretical solution. The research method used at this step is Applied research since the aim here is to find a concrete solution to a practical problem, utilizing existing research at times.[9]

2. Then, a prototype is developed for a tool which will be use-case for the evaluation of the solution. Developing the prototype takes place in a phased manner. ‘Phased’ here refers to implementing series of solutions one by one, but implementation of next depends on successful implementation of the previous. Every phase requires achieving a certain objective in terms of developing a technical product. Hence, we use agile software methodology. At every step in the development of generalized approach, a technology is applied to solve the problem at hand. This technology is selected after carefully evaluating the pros and cons of all the technologies available. Required data and use-case knowledge is collected from existing systems, and if needed, via semi structured employee interviews (and utilizing laddering technique [7]).

3. The next chapter will focus on validating the solution by applying it on a tool that acts as a use case and verifying if the performance is as desired. Agile software development methodology is the methodology with shorter sprint release cycles rather than the long release cycles that used to occur in the waterfall techniques that were earlier used. [34] This helps in shaping the product according to the requirements and especially when the requirements might change rapidly or might have to be adjusted according to the customer needs. The approach is still an iterative approach, but helps all the developers and stakeholders to estimate the schedule of the remaining software development process in a more comprehensive and accurate manner. [34] The whole process is focused on involving the stakeholder or the customer in the software development process by constantly demonstrating the functioning product to them and then changing it or further develop it according to the input given by them. It involves identifying the stakeholder, then prioritizing the stakeholder according to the influence they have on the whole product and finally, communicating with the stakeholders with the intent to involve them according to the priority list. [34] Also, the stakeholders should be contacted in their preferred way of communication. The whole idea behind the agile software development approach is that the complete set of requirements can be broken down into smaller more simpler requirements. This way the developers can easily estimate the time required to finish the development of the remaining product. Hence, to sum it all up, one can say that the agile software development methodology rests on two principles: one of shorter release cycles and the other of more frequent and closer stakeholder involvement.
3.2 Developing the Automation Approach

In the succeeding sections, we discuss the automation of the OSLC adaptor generation. However, it should be kept in mind at all times that we aim to develop an adaptor that would work as an OSLC service provider. We do not generate the code for that part of the adaptor that acts as an OSLC client. Once the approach is validated for an OSLC provider, the generation of OSLC consumer with the same approach automatically follows without considerable challenges.

3.2.1 Step 1: Mapping relational data to RDF

The first task in order to develop a generalized approach to automate the development of an adaptor for a tool is to get the information about the metadata of the database residing inside the tool. As we have seen in the previously developed adaptors, once the metadata is available, it can be used for designing services offered by the developed adaptor. The adaptor services designed with the metadata are for providing access to the data inside the tool. With the help of this service, any other tool can request data from the tool database regardless of its technology or platform. In addition to this, these services can also be used by other tools to create new entities inside the database or to insert new data inside the database of the tool via the adaptors. Also depending on the type of access provided, data inside the tool can also be removed when needed. An important thing to keep in mind here is that the database referred in this scenario is always an SQL database and not of any other type. Now, there are several methods via which the metadata information can be read from the database. The method that needs to be adopted depends on which form of data will make the whole process efficient and easily replicable since the aim is to develop a generalized approach or algorithm. To achieve this, we need the metadata that makes the succeeding steps in the approach swift and efficient.

One of the most common way to obtain the metadata of an SQL database is to get the database schema as an SQL query file. This can be done by using various tools developed like ADO.NET or it can be done by running SQL commands which are provided in every common DB system. You get database schema in the form of scripts. Metadata can also be directly obtained inside a Java or a .NET program using numerous methods like ‘getSchema’ in the connection class of the .NET framework. [33] Apart from these, there are several other approaches that obtain data and metadata from database in the form of RDF. This way of obtaining data and the metadata of a database uses the concept of converting the data into triples primarily RDF triples. These techniques use RDF to integrate data coming from different sources. As described in the paper by Kate Byrne [44], where the conversion of RDB(Relational Databases) to RDF is discussed in context of cultural archives, RDF puts more data in the reach of the public web querying, links separately curated data and representing associated multimedia archive that is not efficiently retrieved when stored in RDB.

In the survey [43] that evaluated the existing ways of mapping relational database to RDF done by the W3C RDB2RDF Incubator Group, focus is to analyse the information gain that occurs by mapping RDB to RDF in different techniques. The authors categorize and then analyse different techniques on the basis of 6 parameters that they define. First of the parameter is how these mappings are generated from RDB to RDF form. For example these mappings might be generated automatically where an RDB record is mapped as an RDF node, the columns act as the links between these nodes and the table values are treated as RDF literals.[43] Another example is where these mappings are semi-automated with domain ontologies being explicitly modelled. These domain semantics give more meaning to the
converted data which helps in increasing the amount of data accessed via queries. When used, the domain ontologies also help remove redundant triples thereby reducing the size of the whole RDF data created. Now these ontologies might be user created, focussing on the domains that have not been used in RDF conversions yet or might be pre-existing. An example of such a predefined domain ontology discussed is the hydrology ontology used in mapping spatial data to RDF. [45] Second classification is based on how these mappings are presented and how the generated mapping can be accessed. [43] The aim here is to make the mappings accessible as easily as possible so that it can widely be used accelerating the use of RDF technologies. The next and the most important classification is whether the mappings are generated dynamically or statically. The static approach is an ETL (Extract Transform Load) approach wherein the conversion tools read and convert the RDB data into RDF data and create a dump. This happens like a batch process which might run at regular intervals. A query made by someone to access RDF data is run on this dump. The actual data in the RDB might have changed since the dump was last generated and hence the mapping available via this method might not be the most recent. However, it holds some advantages as well. Once the dump is generated, extra information may be inferred from this dump by using inference rules that may be defined by the user or by a consortium like W3C. This can be done while simultaneously maintaining a low query execution time. A dynamic approach however runs the query directly on the actual data being stored in the database either by converting the query or the data in the database to a compatible format. This way a more real time picture of the stored data is accessed via the query. However, an inference of the RDF data might not be possible or might be time consuming and may increase the query time considerably. As hinted previously, another classification may be created on the basis of when a query is executed to retrieve data, whether the query is converted to a form compatible with the database or the query remains unchanged and is executed on a converted form of RDB which is compatible with the query. The paper however also lists some other classifications which might not be of significant relevance in our context. These classifications help us select the best tool or technique that we can use according to our needs at this step of the process. There are some other issues that need to be kept in mind when converting relational data to linked data. One of the primary one is differentiating private data from public data. There is often no way to control which data is being published or is exposed. This deters the spread of linked data as a mainstream technology. Another obstacle in the use of linked data is mapping the relational data with the domain related ontologies. Most of the time this happens due to insufficient description of the database schema. For example, insufficient description of foreign key constraints in a database. This might happen more often than you might think since many databases develop overtime and don’t enforce strict rules. Access to mapping file that is generated for a particular database is also important. By accessing this mapping file, we can edit it to cater to our needs and how access is given for the data inside the database. We need to extract the schema of the database for which we are building the adaptor, in RDF form. This schema can then be fed into subsequent tools automatically that further the automation process of the adaptor. It is also preferable if the conversion tool provides with an option of converting RDB data to RDF data both dynamically i.e. on demand in response to queries and statically in batch mode.

Many of these problems need to be addressed in the tool that we select. Or the tool should be able to support the resolution of these problems in any other way possible. Hence these play a pivotal role in selection of the tool for this step as well. The following section discusses some of the most prominent tools and techniques which are available for our use, the one which is finally selected to be used in the approach and why the approach has been selected.
3.2.1.1 Virtuoso
The first technique available at our disposal is known as *Virtuoso*. It is a tool developed by OpenLink Software that acts both as a database (relational or federated) and applications platform. [47] The Virtuoso database stores data in the form of triples. With Virtuoso, one can access data inside Virtuoso’s database or another relational database in a desired format. Virtuoso is used to generate automatic mappings between relational data inside a database to RDF data. Virtuoso provides virtual RDF graphs from RDB data without the creation of physical RDF datasets [43]. For this, it uses the general rule of automatic mapping generation wherein the primary key of a record is treated as an RDF object, the column names act as predicates and the column values are RDF subjects. Every RDB table is treated as an RDF class node. Virtuoso supports numerous other standard such as HTTP, XML, JDBC, SQL, etc. to name a few along with all the RDF data types. Virtuoso brings together data from different databases, which might be storing data in different formats or might be running on different platforms and vendors, to answer the queries made by web, local tools, other applications, etc. Virtuoso pretty easily creates XML data from HTML documents, SQL databases or syndicated XMLs. [47] Virtuoso also provides with a transitivity feature used for applying graph algorithms for inference purposes. Another great advantage of Virtuoso is that it converts data to XML format dynamically from heterogeneous or homogenous databases of different types when responding to a query. The data conversion hence takes place on the fly. Virtuoso RDF views provides with its own ontology language. This is Quad map language that provides the functionality to define RDF views on relational data. It means that Virtuoso stores RDF data in form of quads which are just like RDF triples with just one additional element. This element is known as ‘Context’ and is used for naming the RDF graph, in order to sustain the provenance of the RDF data in the graph. It is a meta-schema language that is similar to SQL DDL syntactically. The tool also provides with a SPARQL endpoint for the data. The tool provides with data access drivers and tools to help with applications development, deployment and other integration challenges. Drivers for Sesame, Jena and Redland frameworks are also provided.

Virtuoso however, provides no option to edit the quad map language that it uses to map RDB to RDF according to our needs. Nor it generates an intermediate mapping file that we can edit to control what data is given access to. Most importantly, the mapping file would have given us access to the schema of the database in RDF format. We need this schema to automate the adaptor generation process. Although the fourth element in the Quad, i.e. the ‘Context’ is really helpful in keeping the provenance of the RDF data, since with the onslaught of too much RDF data being published online today it is difficult to maintain the quality of that data. Also, the tool does not generate triples on a schema level. Though not of relevance here, the mapping language used in Virtuoso does not provide RDF based write support as well.

3.2.1.2 Triplify
Triplify is another approach to convert relational data stored in various relational databases into linked semantic data. The Triplify is an open source R&D project that was launched by the research group AKSW (Agile knowledge Engineering and semantic web) hosted in University of Leipzig [49]. Triplify tool has been coded in PHP programming language [48]. The underlying concept behind Triplify is to map the HTTP-URI requests coming in and requesting data to relational database queries. Once the data is retrieved, it is then converted into RDF statements. These can then be published with any desired format, the most useful of which is RDF serialization formats including, but not limited to JSON, CSV, etc. Hence, linked data can be retrieved from a relational database.
One of the major benefits of the tool is its lightweight. It is easily deployable and integratable with already existing web applications. Another added benefit is that it implements incremental crawling of linked data sources. This enables data holders to publish update logs of only that linked content onto web that has changed and saves the effort of going through the unchanged content and re-publishing it needlessly. These update logs are also published in the form of linked data in order to make it understandable to web crawlers. In addition to that it comes with configurations that contain already configured mappings for many relational schemata like WordPress and Drupal [48]. Inspite of being lightweight, Triplify can be used on datasets which are considerably large in size as well, for example geo data from OpenstreetMap Project which is a hefty 160GB large, as described in the work of Auer et al. [48]. Triplify doesn’t mandates the use of a mapping language. One of the implementation advantages of this technique is that it produced RDF data from relational data both on demand dynamically and also generates RDF dumps of the data, also known as ETL technique. This enables us to integrate it into a server to provide on demand RDF data from the database in response to a request while on the other hand, inference rules can be run on the static dump to derive even more information from the database. It also enables the integration of data from different sources due to the use of same configuration, without even mapping to an existing vocabulary. [48]

Now, one of the drawbacks of this technique is that it employs SQL as a mapping language [48]. The reason for this, as given by the authors of the technique, is the level of maturity of the SQL language and the support available for it. This pushes the mapping task to the database system and brings down the effort needed to perform this task. Although quite flexible, the language is not as focussed on expressing the ontology mappings as some of the other languages, discussed in the proceeding sections. One of the drawbacks of using this technique is that it has no support for SPARQL query language. SPARQL language is one of the most commonly used languages in the context of linked data. And no support for it definitely hinders the use and spread of Triplify as an RDB to RDF technology. Triplify also asks developers to manually remove sensitive data from the SQL queries so that information that is publically accessible is not exposed. This however is inconvenient and impractical when scaling the technique or when being used in an automation process. Triplify does not generate an intermediate mapping file in the conversion/mapping process. This intermediate file could be edited to control what data is given access to as RDF data. This file also provides access to the schema of the database storing the relational data. Hence, there is also no way to get hold of the database schema in RDF format here. The tool also provides only a partial way to publish updates to the database via the update logs.

3.2.1.3 R2O

The next most promising option we have to perform our task is the R2O mapping language. R2O stands for Relational to Ontology language. R2O is an XML based extensible declarative language that can be used to map relational databases to RDF. R2O comes with a set of primitives that have well defined semantics. [46] It is deemed good for map cases that are complex due to low similarity between ontology and database model. The conversion process involves creating a mapping description document that links the components of the RDB with the components of the ontologies being used. This document can be used to populate an ontology with different instances from the data in the database. This is done via batch processing where an RDF dump is generated or can also be done dynamically by translating the SQL queries in real time. Hence, the R2O approach provides both dynamic and static conversion options. The RDF dump can be worked upon to produce more information hidden inside the generated data. Another unique feature of the language is that it can be used to detect inconsistencies and ambiguities in the definition of the
mappings i.e. an ability of self-verification. R2O works on any database that implements SQL standards, independent of the database management system being used. R2O is also capable of automatically characterizing the data sources using the mapping definition thus enabling dynamic query distribution for information integration.[46] R2O has also been used in providing ontology based access to streaming data sources.[50] R2O mapping language can be used in combination with any other suitable tool. ODEMapster is however the most prominent tool that uses R2O as a mapping language to convert RDB data into RDF data. ODEMapster is developed as a part of the NeOn Project and is included in the NeOn toolkit as a plugin. The plugin offers a GUI to the user with which the R2O mappings can be created, executed or queried. [51] ODEMapster is a mapping processor that can be utilized to process the mapping description document generated using the R2O mapping language. It offers two modes of execution. One where translation occurs on the fly in response to a query and the other where translation of the entire data repository takes place like a batch process.

R2O language however is not a very dominant language when it comes to semantic data and has been used in limited number of projects. The R2O language is unable to provide write capability. Thus one cannot write the RDF data into the database via the SPARQL or HTTP requests. While converting the data from RDB to RDF form, neither R2O nor the ODEMapster software enables us to control which data is given access to or is exposed from the database. Another drawback is that the mapping document that is created in the process of conversion is not to be dealt with manually. For all the operations to be performed on the mapping document whether it is the generation of the document, editing the document or even the browsing, graphical user interfaces are to be used. [46] The database schema description generated by the R2O language is in a typical format and is not in a very common format like Turtle, XML, etc. Hence, it is comparatively difficult to read this schema information and use it elsewhere as per our need. Another major disadvantage of using the ODEMapster processor is that it is no longer being developed and maintained. Hence there is no development support available if one wants to use it in a project.

3.2.1.4 The D2RQ Engine (D2R Server)
D2R server is again a tool that is used to convert relational data to semantic data. The server is developed as a part of the knowledge nets project being carried out in the InterVal-Berlin Research Centre. [52] D2R server comes with a lot of features and is part of the complete D2RQ platform. Figure13 [53] describes the complete architecture of the D2RQ platform. It shows the position of the D2R server and its surrounding components. A D2RQ engine that operates on the D2RQ mapping language interacts with the RDB directly. It then creates an intermediate mapping file that consists of directions on how to map the relational data elements to the vocabularies and ontologies being used. The D2RQ engine then utilizes the generated mapping file to map the relational data into RDF resources. It has an option to generate an RDF dump, which is a static means of generating the RDF resources. Or another way is to use Jena API to access this data. Both these ways are to access the data if the D2RQ engine is running locally. In order to make the data available remotely or to publish them on the semantic web, we use the D2R server. The D2R server, as visible in the figure, provides 3 end points. It provides one SPARQL endpoint for SPARQL clients to access the data, one RDF endpoint for linked data clients (which could be an RDF browser) to access the data and one HTML endpoint to serve data to HTML browsers.
D2R server uses a mapping language known as D2RQ mapping language to identify and generate resources and properties from database content. D2RQ is a declarative mapping language which is mainly used to map data stored in relational databases to RDF vocabularies and OWL ontologies. This language describes the relation between the data in relational and the RDF form. [56] The mapping is written in turtle syntax and is stored in the form of a RDF document. D2RQ namespace is utilized to express these mappings. [56] The namespace terms can be found in the D2RQ RDF Schema [57]. The D2RQ mapping language converts the data inside the database into a virtual RDF graph which is analogous to the concept of views in the relational data world [56]. The main entity of the D2RQ map is the ClassMap. ClassMap is the mapping of all the major entities in the database to a class or a group of classes of resources. Now each of the ClassMaps consists of PropertyBridges which describe the creation of resource descriptions. The language is of course used in the D2RQ platform. The D2RQ platform is compatible for a large number of different databases like Oracle, MySQL, PostgreSQL, SQL server, HSQLDB, etc. [58].

D2R server complies with the principles of the web specified in the W3C recommendation made by Ian et al. [54] First of which is that Important entities are referred to as resources and are represented by URIs. Consequently, dereferencing these URIs gives these entities represented in a form dependent on the content negotiation of the internet media types. D2R server maps not only the contents of the database into the RDF format to be presented onto the semantic web but also makes it possible for a user to browse the data stored in the database in RDF format and also to search through this data. These functionalities are the two main paradigms of accessing the semantic web. [53] Users are able to browse this data using there RDF or HTML browsers. If one wants to query the database, it can be done using the SPARQL query language which is based on the SPARQL protocol. These queries are transformed into normal SQL queries by the D2R server that has been implemented. And these SQL queries are then run on the data to fetch the desired results. Once these queries return the result, the server again transforms them into the desired format, as requested by the user and are then rendered to the user. The data might be rendered in the form of RDF/XML format or might be converted into HTML and will be rendered as an HTML page. The results are also obtained in XML format and SPARQL/JSON serialization. So the task of
the D2R server among other things is to map data from relational to RDF, allocating a URI to the data stored in the database so that it can be accessed like any other resource on the internet (referenced with a URI) and dereferencing the URIs. Another task that the D2R server does is to perform hyperlinking. Since D2R server uses the mapping language D2RQ, the developers of the server have also developed a tool inside the D2R server that uses this mapping language to generate D2RQ mappings from the table structure. These D2RQ mappings play a vital role when data is accessed from the database. They specify rules on how resources need to be identified in the database. They also say how the property values can be generated from the database. With this tool, a new RDF vocabulary is developed that is specific to the database whose data needs to be exposed. In this vocabulary, the table names from the database are used as class names while the column names are used as property names. The following figure 14[56] shows a D2RQ map. The ClassMaps here are ‘Paper’ and ‘Author’, which are entities in the database. They further have properties that are described by a PropertyBridge.

Figure 14: Structure of a D2RQ map created from a database table. [56]

A ClassMap specifies how URIs will be generated for different instances of the classes. The class maps generated with the server contain URI patterns which are used to assign URIs to entities inside the database. Say a pattern like: ‘students/student@@student.Number@@’ will generate a relative URI that will look like ‘students/student338042’. This relative URI is generated by replacing the ‘number’ word here with the actual student number from the database. This URI is then assigned the final form by appending to it the base URI of the server. This way the HTTP requests coming to the server for these URIs can be answered by the server by dereferencing the URIs. [52] However, if the database content comprises of URIs, then there is no need to create artificial URIs for such content. Since in that case these actual URIs stored in the database can be used to identify this content. An example of such a database is one where a lot of web documents like HTML are stored. Now when retrieving RDF or XHTML representations of the resources, an HTTP request needs to be sent to the server. The server will then dereference this URI to find out which resources are being asked for. The server then retrieves the resource and presents it to the user in XHTML format if the content type requested in the HTTP request was text/html or application/xhtml+xml. On the other hand, if the HTTP request has a content type of application/rdf+xml, then the server
produces an RDF representation of the resources.[52] If the resource being requested by the HTTP request is just an information, then different representations can be served as a response. However, if the resource requested is a real-world entity like a person, then the server produces a response which is nothing but a second URI that points to this resource. When this second URI is queried, a description of this resource is served. This behaviour implemented by the D2R server was first suggested by the technical architecture group (TAG) part of W3C.[55] The server also inculcates hyperlinking when producing the XHTML and RDF representations. Many of the RDF triples generated by the D2R server have a URI link as their object. This concept helps link together the resources produced by D2R with other database elements and other RDF documents on the internet. Another feature of the D2R server is that whenever it generates the RDF representations (documents), it adds to the resource description an rdfs:seeAlso triple that points to an RDF document that consists of links to all the other resources that might have been generated using the same ClassMap.[52] Not only RDF level hyperlinks are created via D2R but XHTML representations of these hyperlinks are also present.

One of the biggest advantages of using D2R server is that it employs the custom mapping language D2RQ that is totally customizable. This gives us a real advantage as it can be moulded to fulfil our needs and requirements. This language can also be used in combination with another tool like TopBraid Composer instead of D2RQ to produce exactly same mappings. Hence, though we are using D2RQ, we are not bound to using only D2RQ, in case we need to replicate the process. The D2RQ engine creates D2RQ mappings using the mapping language. These mappings are generated and stored in a mapping file. User has full access to this mapping file and can be customized by the user according to the needs. This file consists of auto-generated vocabulary terms. These terms can be replaced with terms from the prominent and widely used vocabularies in order to make the mappings more widely usable and acceptable. This is again a really big benefit since the user can also use self-developed custom vocabularies. The terms can also be named to something else that the user wants. The names should not be necessarily same as the names in the database. The resources present in this file that represent the database elements can also be removed if the user does not wants to expose certain database parts via the adaptor that will be generated. This solves the research question of managing the privacy of the sensitive database content. This intermediate file is also a schema of the database listing all the tables, primary keys, foreign keys, etc. So we also don’t need any other tool in order to get hold of the database schema. Moreover this intermediate mapping file generated by the D2RQ engine is in turtle format. As discussed in previous chapter, this format is very common in linked data community and widely used. This file can hence be further worked upon very easily. So much flexibility with this intermediate file is really advantageous since this file can and will be used in the further process, as an input to the next step of the algorithm. Another advantage that other tools don’t offer is the mechanism used to create the URIs for a resource inside the database. This mechanism ensures that the identifiers belong to only the URI space that is owned by the server itself and help dereference the URI and obtain the information regarding the queried resource form the URI itself. The functionality of hyperlinking implemented by the D2R server also helps in crawlers and RDF browsers to browse through the semantic data produced by the D2R server easily. This helps in doing comprehensive analysis of the linked data by data analysis tools and frameworks. Another contrasting feature of D2R server is that the D2RQ platform used in the D2R server gives access to relational information inside the database via Apache Jena API as well. Along with that, just like some of the previous tools, it provides the option of creating an RDF dump which is static. It might not present the database dynamically, but it provides an opportunity to apply inference rules and extract more information out of the already available data. On
the other hand it provides the opportunity to query the data dynamically so as to present the current picture of the database.

The number of ways to access the relational data offered by the D2R server (the D2RQ platform as a whole) are the most among any other previously discussed techniques. One can access the data using SPARQL queries, via HTML views of the stored data, as a linked data server, taking linked data dumps of the data viz a viz the ETL technique and a Jena API access to the database. Due to all these overwhelming reasons, D2RQ engine becomes the first choice to be used in the process.

3.2.2 Step 2: Creating the Adaptor model.

As explained in the previous chapter, a meta model for the adaptor is like the description of what data will be contained inside the tool for which the adaptor is being developed. Along with that it also describes the functionality that the adaptor will provide. This metadata helps other tool-adaptors to know all this information, when they get hold of this metadata and parse it. This model based approach of developing the adaptors helps in discovering the adaptor and also in using the functionalities it offers in the form of the RESTful services. [35]

Hence, we now need to convert the database information obtained into a metamodel for a to-be generated adaptor.

The database information that we need to obtain from the database mainly consists of the database schema along with some other information. Most of it can be extracted from the intermediate mapping file obtained in the previous step by using the D2RQ engine in the database. Step 2 can be divided into 3 sub-steps, which when successfully carried out take us to step 3 of the approach.

3.2.2.1 Reading the resources from RDF model in Turtle File

The first sub-step involves reading the intermediate mapping file generated by the D2RQ engine and obtaining the database schema so that it can be used further. The mapping file generated is of the format Turtle (.ttl). As explained earlier, its structure is a set of triples where the subject of these triples are the table names and the column names while the object of these triples are the properties relating to the tables and table column. It also consists of other information like Join operations on the tables, foreignkeys and primarykeys among other things. Now to read these table names (can also be referred to as resources) and columns (can also be referred to as resource properties) and other details given in Turtle RDF serialization, we make use of the Jena Model API. With Jena, one can create and read RDF data from RDF files and graphs using the core API. One can also serialize the triples using RDF/XML or Turtle formats. [59] Specifically, the core RDF API of the Jena can be interacted with, in order to read data from this turtle file. Jena considers that all the information written inside the turtle file is stored in a data structure known as a Model. The data inside the file consists of RDF nodes connected together with labelled relationships, where the relationship is unidirectional. Hence we can say that the term Model is used by Jena for an RDF graph. The Jena API functions in combination with Java programming is used to read data and then persist it into the meta-model. Hence, this Model is considered as a class and container containing all the information stored in the form of the RDF graph. The Model class is defined with numerous methods that help read and write data to the RDF graph inside the turtle file. This also makes it really easy to write programs and applications based on RDF. Another simpler abstraction level is Graph API. It is easy to re-implement for different RDF stores and acts as a common interface to them, specially the RDF stores at a lower level. Hence, we can say that Jena has 3 different concepts of the RDF container. The first is the conceptual graph, which is nodes connected together with labelled relations. The
second concept is *Model*, that is useful from developers’ perspective. It is a Java API with numerous methods that can be utilized by application developers. And finally *Graph API*, which is a simpler Java API that helps extend the functionality of Jena.

### 3.2.2.2 Combining the read data with a Vocabulary

So after reading the resources from the turtle file using the Jena API and Java programming language we proceed towards the second sub-step. The second sub-step involves taking a decision about which vocabulary to use, with which the read resources can be combined and linked in order to give them a more general meaning. We choose to use the OSLC specification in order to give a more generalised meaning to the resources identified so that they can be used for semantic exchange via the generated adaptor. As already discussed in the previous chapter, the reason for this decision is due to the fact that OSLC works specifically towards integrating industry tools and develops specification that are custom made for the industry tool integration scenarios. In this particular thesis, we use the OSLC Core Specifications, OSLC Architecture Management Specifications and OSLC Change Management Specifications. OSLC core specification and OSLC Change Management specification has already been discussed in detail in the second chapter. The OSLC Architecture Management Specification is developed by the Architecture Management Specification Workgroup. The specification defines HTTP based RESTful interfaces in terms of HTTP methods that takes care of different scenarios such as the management of product design artifacts and also related resources such as change requests, requirements, etc. [60] The group defines the specification by finding out a least common denominator that can be managed by AM server and be useful to the goals of Architecture Management in collaborative ALM environment rather than redefining model storage formats or the resource notations. [60] The Architecture Management resource properties are not just the one defined in the specification by the group and can be extended by the user. [60] However, a user shall not define new terms with meanings same as that of already existing terms in the specification. The OSLC consumers and service providers implementing this specification should be compliant to both this specification as well as the core specification of the OSLC core. Further ahead in the process, we would need to use the Lyo code generator for the automatic generation of code for the adaptor. It generates the code from an adaptor model, that we are currently trying to develop. The adaptor model in nothing but an instance of the meta-model persisted with data. The need for a meta-model is fulfilled by the OSLC4J meta-model. For this reason, we use the OSLC4J meta-model. Hence, it is also imperative to understand the OSLC4J meta-model. As described in figure 15[61], the meta-model consists of 3 major parts. The first one is the modelled OSLC domain specification that are either consumed by the adaptor or exposed by it and as defined in the OSLC core specifications defined by the core working group.[61] This has AdaptorInterface instance at its root. The instance consists of domain specification and namespaces with no other properties. Further, you can create a DomainSpecification instance as a child of this instance. This represents the domain we are working with for example Architecture Management (AM) in our case. Further you can create NamespacePrefix as a child as well. Further one can define properties for all these instances. Some other required instances are for example Resource and ResourceProperty instance. [61] The Resource might refer to the ResourceProperties in this Domain Specification or from any other Domain Specification within the model. The second part of the meta-model is the Server. This involves modelling the services that the OSLC adaptor will be exposing. Under this part, you define the properties of the AdaptorInterface instance defined in the previous part. Example of some Properties are Name: which contains the name of the adaptor, Java Files Base Path: where the code for java files will be generated, etc. We then create ServiceProviderCatalog instance as child for the AdaptorInterface. All its properties can be filled in as well. Then children of
ServiceProviderCatalog can be created as Publisher instance and ServiceProvider instance. To control the relative URL of the ServiceProvider JAX-RS service, we can define the property serviceNameSpace and instanceID. From each of the service providers, we can then create Services and Capabilities instances. We then fill in the properties of the instances that we have created. In order to control the relative URLs that are used when handling a resource, we can create a BasicCapability instance. The third part of the meta-model is the client part. This involves modelling the interactions that the adaptor will do with a Service Provider Adaptor as a client. [61] For this, the first instance is ModelledRequiredAdaptor which is created as a child of the AdaptorInterface instance. In its properties, we connect this instance to the adaptor model that client represents. We also define the URI of ServiceProviderCatalog here.

Now, we need to generate java classes for key concepts described in the OSLC Core and other domain specifications and for the meta-model for OSLC4J code generator so that data can be persisted. For this we use what is known as the Eclipse modelling framework (EMF). EMF is framework that enables us to generate code for any tool or application based on a well-defined and structured data model. [62] This data model can be expressed in an Ecore or via any other means. The whole EMF rests on 3 pillars. The most important of them is EMF core. This comprises of a meta-model (Ecore file) that acts as a description of the meta-model and a reflective API that can be used to manipulate EMF objects. The second pillar is the EMF.edit framework that includes reusable classes which are generic in nature which build editors for EMF models. The third and the final pillar is the EMF.codegen. This is the part that supports generation of code and everything else that is needed to build a complete editor of an EMF model. It comprises of a GUI to invoke code generator and specify generation options.[62] EMF core and the EMF.codegen are of vital importance to us when generating the code for the meta-model instances and later for the adaptor.
As discussed just now, EMF provides us with a GUI to model all the instances along with the properties, for which we need to generate the code. Figure 16 is a snapshot of the GUI tool that the eclipse provides in order to model all the entities of the meta-model in figure 15.

Figure 15: OSLC4J Meta-Model
This GUI can be used to create a model (the ECore File), edit the model and add more instances to it very easily. This option is in addition to a command based editing of the model. [62] Once the Model is created using this GUI, we execute the tools and related run time support provided by the EMF to generate the Java classes and other related resources corresponding to the meta-model. Figure 17 is a picture depicting the generated java classes in eclipse from the OSLC4J meta-model using EMF.

Figure 16: Eclipse provided GUI to model instances graphically.

Figure 17: Generated Java classes corresponding to the meta-model.
3.2.2.2 Persisting the read data to a Meta-Model

The final adaptor meta-model file with persisted data, that we need to obtain is of the form .XMI and consists of the persisted meta-model. XMI stand for XML metadata interchange and is a standard defined for the exchange of metadata information via Extensible Markup Language (XML). To obtain this file that is the persisted meta-model, we need to link the data that was read using Jena, with OSLC specification and the java classes corresponding to the OSLC4J meta-model that have been generated using the Eclipse Modelling Framework (EMF) in the previous step. To link the data with the OSLC specifications and the generated Java classes we again make use of certain other functionalities of the Eclipse Modelling Framework (EMF) known as the EMF persistence framework [62]. This part of EMF provides us with certain Java classes that have the functionality that enables us to link the data with OSLC and write the whole thing into an .XMI file. The OSLC specifications, that need to be linked to the data and the meta-model, are also saved in an .XMI file. The software developed as part of this thesis is written in Java. It fetches the data read by the Jena framework and uses the functions and libraries provided in the EMF for persisting the model objects, to link this data with the java classes and the OSLC specifications. It then uses some more EMF functionalities to write this persisted data model into an .XMI file. This .XMI file obtained consists of the schema of the database along with proper OSLC specifications and is used as an input for the next step.

3.2.3 Step 3: Automatic Generation of the adaptor code using the Adaptor model.

The next and the final step of the process i.e. to generate the final code of the OSLC adaptor uses the Eclipse Lyo project which is an open source community that provides support for the eclipse community to help adopt OSLC specifications and build OSLC compliant tools. Eclipse Lyo consists of 3 major parts. An SDK that helps write Java code for OSLC enabled adaptor that act either as clients or as service providers. This part also known as OSLC4J provides annotations to facilitate Java objects with OSLC attributes. These annotations also assist with resource preview user interface. [39] It provides libraries to develop OSLC service provider and consumer along with resource shape documents. It also contains sample applications and test for these applications similar to Lyo OSLC test suite. OSLC4J has 3 components, a core component that provides annotations and model support, an Apache Jena provider and an Apache Wink Provider. [39] The OSLC4J that generates the code for the adaptor utilizes the project Acceleo. It is an implementation of the model to text Language (MTL) standard which is developed by the Object Management Group MOF. [63] Acceleo help make a code generator by providing the functionality to read any given model and then create Java or other required code according to the specifications given in the model. Acceleo helps Lyo handle the life cycle of code generation as well. It provides with a simple IDE which is easy to use and the syntax to generate the code from the model is quite straight forward as well. Along with an editor, it also provides a debugger, a profiler, a traceability API which highlights the input coming from the model and the parts of the generator that are involved in generating code for this input. Lyo OSLC test suite is suite of tests that make sure that the OSLC domain provider implementations are compliant with the specifications. The suite is based on JUnit. It contains tests covering almost all the major domains like Quality management, requirement management, etc. At the end of the test it provides the developer with a report about the successful execution of the developed adaptor. The third part of the Eclipse Lyo project is the Reference Implementations for OSLC (RIOs). These implementations help users understand OSLC works and also provides with a basic minimum server to test against. [39] Eclipse Lyo uses the OSLC4J meta-model to define the code for the OSLC adaptor.
Hence the Eclipse Lyo code generator takes care of both the OSLC core model (the left part of figure 15) and also the vocabulary i.e. the domain specifications(right side of figure 15) when generating the code for the adaptor. The persisted meta-model stored in the form of .XMI file which is obtained from the previous step is fed to the Lyo code generator as input. It then generates all the Java classes and other implementations that are needed for the adaptor to run.

These steps when implemented successfully result in the generation of the OSLC enabled adaptor that can be used as an OSLC service provider for a tool present in the ALM/PLM toolchains.
Chapter 4

4. VALIDATION AND PERFORMANCE EVALUATION

We have now developed the approach to automate the generation of an adaptor that is OSLC enabled, for any tool that is a part of an ALM/PLM toolchain. However, the accuracy and validity of this approach still needs to be tested. This can be done by implementing the approach for the development of an actual adaptor for an ALM/PLM tool. And then doing a test run of this tool to see if it actually exposes the data and functionality from the tool for which it was developed and check the performance of the developed tool. For performance evaluation, we will also employ the method used by Matthias Behl et al. [36] wherein we calculate the lines of code that were automated as a result of this approach and saved many man-hours.

4.1 Use Case: Sesamm Tool

In order to test the validation of our approach and see its performance, we implement the developed approach on a tool called Sesamm. The tool is an internal tool of the company Scania AB, where the thesis is being carried out. Sesamm tool is developed at Scania itself by the research and development department called RESA. Scania is focused on development of mechatronics and vehicle systems. These systems are made up of numerous individual functionalities. Each functionality is referred to as a UserFunction(UF). UF is described based on the User Function Requirements (UFR). Its implementation is taken care by several ECU systems. An ECU system refers to an Electrical Control Unit system. A user function can be subdivided into Use Case (UC). Sesamm is a desktop application that is used all across the organisation by all kinds of employees ranging from system architects, to product developers and testers. They use it to model the system functionalities i.e. User Functions (UFs). Figure 18 describes the Graphical User Interface (GUI) of the tool that is used by the users of the tool to interact with UserFunction(UF) and other elements of the Sesamm database. The tool also has a connected database that stores all the information. The GUI enables the users to fetch data from the database and create new objects in the database. The database which is present is a relational database. Hence, a perfect pick to test our developed approach. With our approach we will develop an adaptor for the database of the Sesamm tool that will expose the data inside this database and will also provide the functionality that this tool offers. The relational database is hosted on a Microsoft SQL server and we are able to edit the data via Microsoft SQL server Management studio. The option to search through the database via the GUI is under development right now. Some of these tables that form the UserFunction on the Sesamm GUI are namely ‘dboElementAnchorSet_UserFunctionAnchors’, ‘dboElementMetaSet_UserFunctionMeta’ and ‘dboCrcFieldSets’. The aim of the validation will be to obtain the content of these tables from the database via the developed adaptors so that this data can then be combined at a later stage to recreate the UserFunction that is visible via the Sesamm tool GUI. The task of combining the obtained backend data from these various tables that form UserFunction view in the GUI will fall under the category of future work for this thesis. But if we could only obtain raw data stored in these tables via the developed adaptor, it will be a proof of concept for us that the developed approach successfully exposes the data and the associated functionality of a tool and its database.
4.2 Adaptor Development

To start with the development of the adaptor for the Sesamm tool which is our selected use case tool, we refer to the approach explained in the previous chapter. A software is developed, as a part of the thesis that combines all these steps programmatically in order to automate the whole process of going from step 1 to step 3 discussed in the previous chapter. Due to the privacy clause in Scania AB, the company where thesis work is carried out, the software code is not produced in the thesis. The software functionality however will be discussed in details in the coming sessions.

4.2.1 Obtaining the mapping file.

The first step involves using the D2R server to obtain the intermediate mapping file that will help us understand the schema of the database along with other details like the foreign key relations and the joins implemented in the database. The D2R server converts the relational data into RDF data. It stores this data into the mapping file obtained in the Turtle format. So as a first step to our approach we run the D2R server on the Sesamm tool. The D2R server maps the relational entities in the database of the Sesamm tool and converts them into RDF resources which are stored in the mapping file. Figure 19 shows a glimpse of the mapping file generated after the D2R server is run on the tool. As visible in the file, the table names take the position of the RDF triple subject (dbo.ElementAnchorSet_UserFunctionAnchor here). The properties of the tables like datastorage type of the table and UriPattern for the table, form the object part of these triples.
The set of triples following just after this set of triple, are the ones where the corresponding table columns are the subject of the triples like $ID$ and $Key$. The properties of these columns for example $column$ datatype, $Join Constraints$ and the table to which the column belongs, form the triple objects. The developed software (as a part of the thesis) uses the D2R server and reads these RDF resources using the Jena API. As visible in this mapping file, the database elements are converted into an interconnected graph of resources.

### 4.2.2 Adaptor Model creation

From the obtained RDF resources that represent the database, we now have to develop the adaptor model. We use the Eclipse modelling framework GUI and generate the classes corresponding to the OSLC4J meta-model. We also develop the .XMI files that contain the OSLC specifications that act as the vocabulary for the RDF resources read from the turtle file. So the software that is written as a part of the thesis, also performs the task of combining the vocabulary definitions with the read RDF resources and uses the Java classes to persist the data into a model(a .XMI file) utilizing the Eclipse persistence framework, discussed in previous sections. The model is an .XMI file. The contents of the XMI file generated can be seen in the appendix (Figure a). The .XMI file contains in textual form, the resources and the resource properties that will be exposed via the adaptor. It contains the OSLC4J model and also describes how resource properties and the OSLC4J model is combined together with the vocabulary to give the resources a more broader meaning and help the adaptor formation. It also contains the mapping between resources and resource properties with the actual database tables and table columns. This file can be edited according to our needs. For example changing a resource name will change the name with which a table in the database is exposed via the adaptor. Or if we don’t want to expose a table via the adaptor, we can delete from this file, all the details of the resource corresponding to this table. Hence, this helps us in handling privacy and data security issues as well. As shown in figure 20, the .XMI
file can also be opened via Eclipse *Sample Reflective Ecore Model Editor*. This gives us the facility to make changes to the model, edit resource details or even delete them via an easy to use GUI. We can control what tables we want to expose to the user via the adaptor. We can also change the name of the tables or the table columns. This gives us the opportunity to control what schema is exposed via the adaptor.

![Diagram](image)

*Figure 20: Adaptor Model (.XMI file) viewed via Sample Reflective Ecore Model Editor*

### 4.2.3 Generating the Adaptor Code

This adaptor model (.XMI file) is then put as an input to the Eclipse Lyo Project. The developed software invokes the Eclipse Lyo functions and feeds the file in as an input. The code that is part of the Eclipse Lyo project is also edited to handle the generation of certain new functionalities that were not present before. The edits enable the Eclipse Lyo to generate the adaptor with Hibernate framework. The Hibernate framework eases the task of fetching and storing the data into the database via the adaptor. The Hibernate framework considers the tables that are converted into RDF resources as Java objects (POJOs). The Eclipse Lyo as described earlier uses the Acceleo Project. The Acceleo code inside the Eclipse Lyo reads this model and performs its operation which is to generate code from a structured model specified in `.XMI` file in detail. The code is generated as a separate Java project within the Local machine. This project can now be run and queried, to interact with the database. This adaptor acts as a wrapper around the tool database handling all the requests coming for the database. The functioning of the adaptor i.e. interacting with the database through it is explained in the coming sections.

### 4.3 Components of the developed adaptor.

The adaptor developed is made up of numerous technologies and concepts. Figure 21 describes the different parts of the adaptor developed. Eclipse Lyo project forms a part of the developed adaptor. Lyo now has an OAuth provider framework. OAuth forms another part of the adaptor that enables the Sesamm tool to act as a provider and is responsible for authorisation. OAuth is an authentication framework that is used by applications to provide limited access to an HTTP service for an external 3rd party application. Who is allowed to access what amount of data that is being exposed, is taken care of by the OAuth. OSLC is
the part that helps exchange data between various tools by exposing tool data according to the specifications. It provides with an option to annotate the POJOs to generate OSLC documents. Some other technologies used in the adaptor are discussed further.

**Figure 21: OSLC Adaptor Architecture**

4.3.1 JAX-RS

REST is a technology that is used while communicating with the adaptor. REST stands for Representational State Transfer. It is a stateless, cacheable communication protocol that is used in a client and server paradigm. REST most of the times uses the HTTP protocol. The requests that the adaptor understand and the responses it produces are REST based. The most crucial part in the development of the adaptor is the use of JAX-RS. The developed adaptor uses the JAX-RS framework Jersey. According to the definition of the JAX-RS framework, it is a Java programming language API that enables the creation of web services that are REST based. As already pointed out in the IBM document on the OSLC adaptors [42], JAX-RS is efficient in delegation of routing different HTTP verb driven requests and the task of serving different contexts. JAX-RS automatically takes care of the responses that the adaptor needs to produce depending on the type of response requested by the incoming request. If an incoming request asks for a RDF/XML type response, the JAX-RS framework automatically converts the data being returned by the adaptor into an RDF response.

4.3.2. Hibernate Framework

Hibernate is an open source Java persistence framework project. For mapping relational data to objects it is one of the most efficient tools. [64] Specifically the Hibernate ORM which enables the persistence for Java and relational databases. It is a tool by which we can work on the database just like we work with objects in Java. Hibernate provides us with its own native API and also provides us with an implementation of the Java Persistence API. [64] We use however the native hibernate API in our adaptor.
The adaptor fetches the data from an incoming request to insert data into the database. It then inserts this data into an object and then with the help of Hibernate, persists this data in the database.

4.3.3 OSLC Annotations

Another important technology present inside the developed adaptor is the OSLC annotations. OSLC annotations are just like any other type of annotations for example Java Annotations. They are specific keywords recognizable by OSLC enabled softwares written on top of Java objects just like the Java annotations. Figure 22 is a snippet of code with OSLC annotations for change management specification [39]. With these annotations, we can identify the OSLC definition, name, property, range etc. of this POJO. These annotations help visually associate OSLC attributes with Java Objects. Not only that but OSLC annotations also help provide preview UIs for the resources. [39] Apache Wink Json4J provider, which is a library developed as a part of the OSLC4J reads the java objects that are OSLC-annotated and helps with their serialization and deserialization.

![Figure 22: Code snippet showing OSLC annotations.](image)

4.4 Functioning of the Adaptor: Validation

The data under the UserFunction(UF) Component which is visible and accessed using the Sesamm tool comes from multiple tables in the database. The data from these tables comes together and is combined to present a UserFunction(UF) in the GUI. The data stored in the database is in a highly normalized form. Now the UserFunction(UF) is formed by a number of tables stored in the database. As a proof of concept we will try and extract data from some of these tables and also insert data into these tables via the developed adaptor which will act as a wrapper to the Sesamm tool.

We start by running the adaptor which was generated as a Java project by the Lyo Eclipse code generator. We also start the database instance which stores the Sesamm database since the adaptor will communicate with this database in order to answer incoming requests. The database for the Sesamm tool resides on a Microsoft SQL server. We will check the updates to the data inside the database done via adaptor using the Microsoft SQL server management studio. As discussed in [35], the approach of service orchestration and service discovery are interleaved. The process of service discovery is automatic. Discovery automatically extracts the details about the tool adaptor service. After the discovery, the client communicating with the adaptor gets only one link (the link to the ServiceProviderCatalog Service of the OSLC core (figure 6)). When the client queries this link, the OSLC adaptor returns a list of ServiceProvider services. When the client queries one of these links in the list, the OSLC adaptor returns multiple links to the various services offered by the adaptor like QueryCapability and CreationFactory, which allows us to work on resources. Figure 23 shows the data stored inside the database corresponding to table dboElementAnchorSet_UserFunctionAnchors.
Figure 23: Data corresponding to table `dboElementAnchorSet_UserFunctionAnchors` in database.

Figure 24 depicts a URL tab where we use a URL (obtained from the adaptor itself) that requests data by calling the `QueryCapability` service of the adaptor. The browser makes a HTTP GET request to the adaptor for data. The URL is dereferenced by the JAX-RS service which finds out which data is being requested. The URL consists of the resource name (`dboElementAnchorSet_UserFunctionAnchors` here) and a primary key name and the primary key value. The adaptor then uses this extracted resource information to get data from the database in the form of an object using the Hibernate framework. Now since the request is coming from an HTML browser, JAX-RS inside the adaptor returns the resource information in HTML format as displayed on the screen in the figure 24. One can see that this information is same as the one stored in the database. (Figure 23) Since `Key` and `FunctionCategoryAnchor_Key` are foreign keys in the database, the adaptor returns them as a link and not as data. Hence, we can traverse from one element to another in the database. Thus our technique has converted the database into a traversable graph in a real sense.
This shows successful retrieval of data from the database via the adaptor. Now, instead of requesting data in HTML format, we can request data in RDF/XML format using a REST client. We query the same URL but set the expected response type to RDF/XML. When the JAX-RS inside the adaptor observes the requested response type to be RDF/XML, it automatically renders the information (retrieved from the database) back in this format. This way JAX-RS takes care of all the different types of response types. Now, just like data retrieval, data can also be saved in the database by calling the `CreationFactory` service in the adaptor. As shown in figure 25, we query the URL by sending an HTTP POST request for Creation of a resource with the ‘content-type’ parameter set to JSON or RDF/XML using Mozilla REST client.
The ‘Key’ value (the primary key of the database) sent to the database is 75978, along with more data to fill other columns of the table. Figure 26 shows the JSON data sent to the adaptor.

```
{
    "@specification/donkeyPrefixes:0:func:CategoryAnchor_Key": {
        "@specification/donkeyPrefixes:0:id": 100,
        "@specification/donkeyPrefixes:0:key": {
            
            "@specification/donkeyPrefixes:0:pref": [
                "http://www.sesamm.com/sesamm",
                "dct:terms": "http://purl.org/dc/terms/",
                "oslc: https://open-services.net/ns/core/",
                "oslc.data": "http://open-services.net/ns/serviceManagement/1.0/",
                "oslc.rdf": "http://www.w3.org/1999/02/22-rdf-syntax-ns",
                "oslc.rdf": "http://www.w3.org/2000/01/rdf-schema"
            ],
            "@specification/donkeyPrefixes:0:type": {
                "@specification/donkeyPrefixes:0:resource": "http://www.sesamm.com/sesamm/dbo/CategoryAnchorSet_UserFunctionAnchor"
            }
        }
    }
}
```

Figure 26: JSON data sent as request

We check the database for the update via the SQL server management studio. We can clearly see in Figure 27 that the update has been successful and the OSLC adaptor has inserted all the sent values into the database.

Figure 27: Snapshot of Sesamm database updated with the data via adaptor

Similarly, data can be retrieved from all the other tables that store the data for UserFunction. All the data coming from different tables can then be combined to form the UserFunction on the GUI. Every table has a specific URL, with which it can be reached. Using this specific URL, data can be edited, inserted or deleted in the table as shown via the example above.
4.5 Evaluation

The primary objective of the thesis was to automate the creation of the adaptor. Hence the true criteria to evaluate the algorithm is analyse the amount of code (Lines of Code (LOC)) that has been generated automatically using the approach. A similar approach has been used by Matthias Behl et al. [36] where the evaluation consists of comparing the LOC generated automatically by the algorithm and the LOC written manually by the developer. So in our case, the total lines of code generated by the automated approach for the OSLC adaptor for Sesamm tool are 35000 (approx.). The line of code that was added manually in order to make the adaptor run is nearly negligible. Due to minor tweaks required to run the adaptor, a total of about 1100 lines of code needs to be added manually. Now, if we pay attention on the ratio of the lines of code generated automatically to the total lines of code (LOC), it is .97 (35000 by 36000), we observe that around only 3% of the lines of code had to be manually added. Rest 97% of the code was generated automatically. This directly converts into saving a large amount of man-hours.

However, the approach of automating the adaptor generations comes with its set of limitations and drawbacks as well. The approach that we have developed, generated an RDF dump of the schema of the database in the first step. This dump (which is a turtle file) is then read using Apache Jena framework. Using the dump implies that the approach creates an adaptor which is not based on the current picture of the database schema. Hence, the adaptor might expose functionality of the tool that might have changed. Hence, the adaptor might have to be regenerated again and again for a tool that undergoes changes quite often. Another point to keep in mind is that the mapping language used for making the intermediate mapping file is D2RQ. Although reliable and used by various conversion engines, this mapping language is not a standardized one. The acceptance of the approach might be wider if a more generalized mapping language is utilized while doing the conversion.
Chapter 5

5. CONCLUSION AND FUTURE WORK

In the following chapter, the conclusion of the work done is presented along with the future work. Also discussed, is the scope of future work.

5.1 Conclusion

The linked and semantic architecture of the resource description framework (RDF) can and is being used to solve a number of interoperability problems. Resource Description Framework and OASIS Open Services for Lifecycle Collaboration (OSLC), which is an open standard in this regard, is proving fruitful in improving scalability and flexibility of databases, which exist in various forms, and communication between them.

A large number of organisations use numerous tools for different phases of application lifecycle management (ALM) and product lifecycle management (PLM) aimed at carrying out activities like requirement management, change management and many more. An example of such organisation is Scania AB which needed to implement the OSLC approach. There are also different departments within the organisation carrying out these activities in parallel, and need to communicate data very often. The main requirement in such scenario was to keep product data consistent throughout the organisation, among different tools and departments. Another task is to perform data exchange or linkage between different tools and departments during the product/Application development process. These tasks become difficult mainly due to the fact that data is stored in different formats and structures or there might be a need to share data without having to maintain a central entity that contains knowledge of the schemas of all the existing databases. The use case in the thesis, Sesamm tool is such an example where data needs to be updated manually if related data in another tool changes.

This thesis developed a solution for the aforesaid tasks by studying all the possible solutions that might prove useful and coming up with a solution to integrate data in different tools for seamless and effective communication. These tasks are accomplished with the development of a tool that will analyse and map the data stored in the databases to RDF vocabulary and present it as an RDF resource. Data in different tools can then be linked together such that data in one tool can be accessed by other tools via REST functionalities utilizing HTML\SPARQL\RDF formats. This is also described in the thesis by querying the database via different methods. It also helps create data (resource) in a tool from another tool (Inter tool communication) using REST. The section talking about the validation of the developed solution, clearly illustrates its success in attaining all the aforesaid objectives. Though working perfectly, other sophisticated approaches might very well exist. Since the data is PLM and ALM related data, using only the concept of RDF and linked data did not suffice. The tools did use RDF but also built on the OSLC framework that already provides with resource definitions (Domain Specifications) in this regard along with guidelines to define further properties and resources (OSLC core specifications). This is by far the most efficient vocabulary to be used in an industry tool integration scenario. It made the work far easier and did not require defining a new industry protocol or ISO standard.
The developed tool and the related research is a step in the direction of bringing the data stored in different relational databases in the reach of one another, without the need of a centralized integration platform, in an efficient way. The developed prototype however tests only creation of new resources in the database along with fetching a resource. It does not implement the functionality to search for a particular element in the database. Neither does it implements the delegated UI dialog functionality specified as a part of OSLC core.

The contribution of the thesis is also towards developing an SQL4OSLC generator [12]. As discussed in [12], Lyo code generator in combination with OSLC4J SDK act as a tool for model based development approach. They work on a specified model without going into much detail about the OSLC standards, etc. They work at a higher level of abstraction. Hence, they need a readymade model (specification). The software developed as a part of this thesis works towards this end. The software automatically creates an adaptor specification from the tool database. And then creates the code that is necessary to access and edit the data in the database on the backend. According to the figure 28 [12], the thesis forms the ‘SQL4OSLC’ part of the top layer which is ‘Technology specific model generators’. The thesis is realisation of the ‘SQL4OSLC’ research part.

5.2 Future Work

The solution developed here is far from being perfect. The first improvement that can be performed on the existing work is to use a more generalized mapping language. An example of such a mapping language would be R2RML mapping language. [65] The language is now a W3C standard and hence commands more impact and support. As discussed earlier, the adaptor model is generated from a static dump of RDF data that is created by the D2R server. This means the adaptor does not works on the current schema of the database. Hence another enhancement to the process will be to develop an adaptor that is based on the most current schema of the database. This process can then obviously be used for dynamic databases where the database schema keeps on changing. These enhancements are a contribution to developing a more generalized approach for automation and contribute to the RDB2RDF community.

Under a more local scope i.e. towards developing an adaptor for Scania AB, a future task will be to combine the data from all required tables in a logical way, to generate the UserFunction view that we see in the GUI of the Sesamm tool (Figure 18) and provide it to the other requesting adaptor. Developing an approach that automatically detects how the Sesamm tool combines all the tables corresponding to UserFunction logical view and makes it possible to render the UserFunction view via the OSLC adaptor will make the approach complete with regards to Scania AB.
References


[8] Fig.3. The qualitative research methods and methodologies. Håkansson, Anne. "Portal of research methods and methodologies for research projects and degree projects." Proceedings of the International Conference on Frontiers in Education: Computer Science and Computer Engineering (FECS). The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp), 2013.


Appendix

A) Adaptor Model (Specifying mapping of table `dbo.ElementAnchorSet_UserFunctionAnchor`