Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelor Thesis

Stefan Greis

Applying Interoperability and Traceability
to the Domain of Testing using OSLC

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Stefan Greis

Applying Interoperability and Traceability to the Domain of Testing using OSLC

Bachelor-Arbeit eingereicht im Rahmen der Bachelor-Prüfung

im Studiengang Angewandte Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr. Zhen Ru Dai
Zweitgutachter : Prof. Dr. Stefan Sarstedt

Abgegeben am 18. Juni 2013

Stefan Greis

Thema der Bachelor-Arbeit
Applying Interoperability and Traceability to the Domain of Testing using OSLC

Stichworte
Open Services for Lifecycle Collaboration (OSLC), Rational Rhapsody Test Conductor (RTC), Interoperability, Traceability, Proof of Concept

Kurzzusammenfassung
In dieser Arbeit geht es darum, Interoperability und Traceabiliy für den Rational Rhapsody Test Conductor (RTC) mit Hilfe von OSLC herzustellen. Hierzu wurde eine Analyse der RTC Konstrukte durchgeführt und ein Mapping zu den passenden OSLC QM Ressourcen erstellt. Der praktische Anteil der Arbeit umfasst eine Prototypimplementierung eines OSLC Adaptors für RTC; dieser dient als Proof of Concept für OSLC Implementierung und wurde unter Zuhilfenahme des Lyo SDK erstellt.

Stefan Greis

Title of the paper
Applying Interoperability and Traceability to the Domain of Testing using OSLC

Keywords
Open Services for Lifecycle Collaboration (OSLC), Rational Rhapsody Test Conductor (RTC), Interoperability, Traceability, Proof of Concept

Abstract
This paper deals with applying interoperability and traceability to the Rational Rhapsody Test Conductor (RTC) using OSLC as a technology. This includes an analysis of the RTC constructs and a mapping to the OSLC resources of the Quality Management domain. In addition a prototype implementation of an OSLC adaptor was created. This serves as a proof of concept for OSLC implementations and is based on the Lyo SDK.

Acknowledgement
Firstly I would like to thank my mentor Prof. Dr. Dai for her time, effort and support which strongly influenced this paper in the most positive way.

Secondly I would like to thank Parham Vasaiely (EADS, MBAT project) for providing me with the topic and the basic idea of the paper.

I would also like to thank Udo Brockmeyer, Marc Lettrari, Ruben Rothaupt and Christian Wachtendorf of BTC for their support regarding the Rational Rhapsody Test Conductor.

I would also like to thank Stefan Paschke for his support regarding the Lyo SDK and its use in the implementation.

Table of Contents

1 Introduction

The complexity of software products has rapidly increased over the past years which also increased the complexity of the software development lifecycle as well. This has led to a vast number of diverse, specific and important tasks which have to be mastered along the software development lifecycle.

Handling those tasks need specialists with a high degree of know-how, expertise and skill. Those experts are specialized in specific tasks and use equally specialized tools to handle the given tasks.

This creates a tool landscape which is highly diverse in regard to the available functionality as well as the number of tools included (see Fig. 1, page 6). Additionally the high amount of tools and tasks lead to a huge number of artifacts that are produced during the lifecycle. These artifacts are not compatible.

Correctly maintaining and handling these artifacts is a complex task in itself. Not being able to efficiently and effectively handle the tasks included in the software development lifecycle will lead to errors which are expensive in both time and money to correct.

One solution is to create proprietary adaptors for every combination of tools (see Fig. 2, page 7). But this is not a very efficient solution as it produces a high number of adaptors which need to be designed, implemented and maintained.

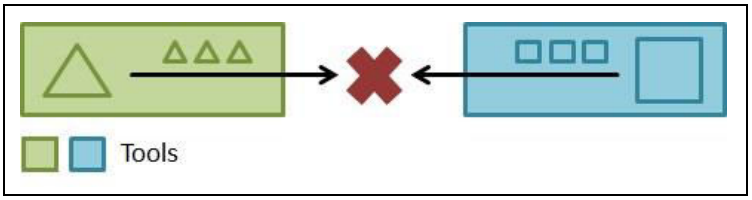


Fig. 1: Lack of interoperability hinders communication.

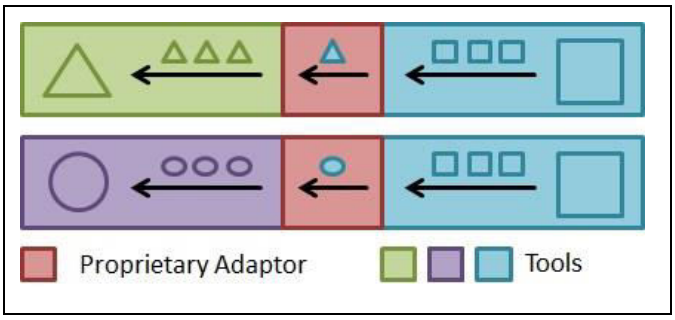


Fig. 2: Point-to-point communication based on proprietary adaptors.

1.1 Motivation

The motivation behind this thesis is improving the tasks in concern of correctly handling different artifacts. This requires the application of concepts and technologies such as OSLC (see Fig. 3, page 7). Applying OSLC to different tools will create interoperability between different tools which will in return make the handling of artifacts along their lifecycle easier. Interoperability is applied as an enabling concept for artifact traceability. Being able to trace the different artifacts is a very efficient and effective way to handle artifacts.

Using OSLC does only require one adaptor for every tool (see Fig. 4, page 8). This is a dramatically reduced number in comparison to the proprietary adaptor approach. Additionally it creates a very flexible way of communication.

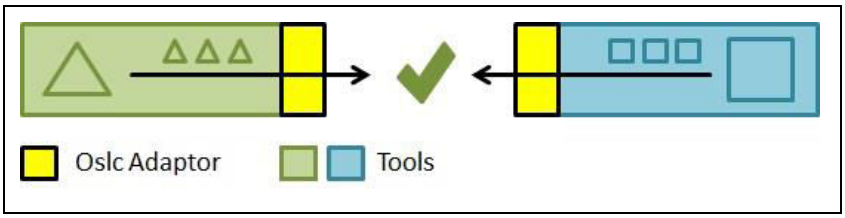


Fig. 3: Communication based on OSLC adaptors.

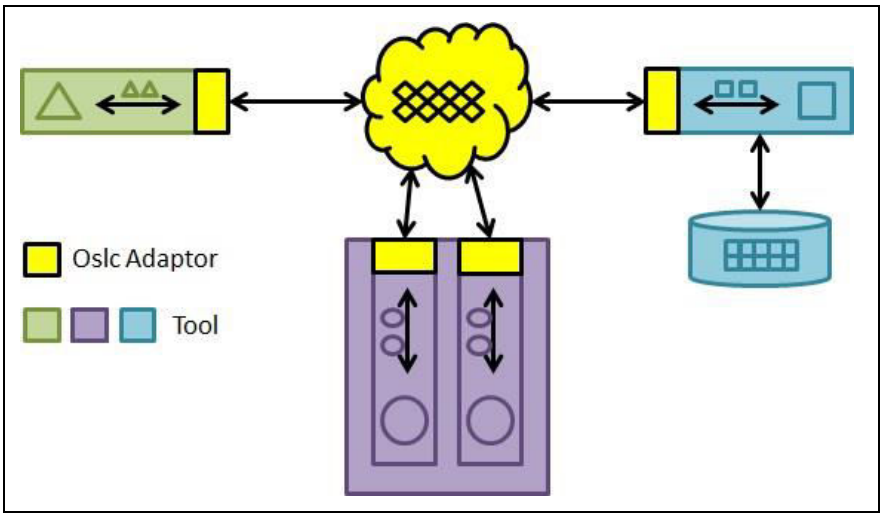


Fig. 4: OSLC adaptors allow flexible communication in a tool landscape.

1.2 Objective

Based on the fact, that interoperability and traceability in the software lifecycle is a very wide area, this thesis focuses on the domain of testing. The objective is to apply interoperability capabilities and enabling traceability with OSLC in this domain. This work will be a proof of concept for OSLC which will include the ability to successfully apply it in theory and practically to a test tool. As example the tool "Rational Rhapsody Test Conductor" is used. The thesis is divided into four parts. The first part gives an introduction to core concepts, core technologies and OSLC. The second part will focus on the analysis and the mapping of concepts while the third part represents the practical side with design and implementation of an adaptor prototype for the above mentioned tool. The final part is the conclusion regarding my findings concerning OSLC.

There are several technologies that can be used for interoperability and traceability but it is not in the scope of this work to list all of them or compare them. Nonetheless there is a short chapter about one of the more well-known technologies to show an example for alternative technologies.

2 Core Concepts

2.1 Traceability

2.1.1 Definition

Traceability can be divided into two basic principles. The first principle focuses on following (tracing) of software artifacts throughout the software lifecycle while the second principle focuses on using the relations, connections and links between artifacts. Traceability is an enabling technology for different tasks in the software lifecycle. An example for such a task is the impact analysis of changes which cannot be done without traceability capabilities. Traceability itself is not a technology but an overall concept which can be applied by using different technologies.

2.1.2 Achieving Traceability

To achieve traceability one or more enabling technologies need to be applied. There are several different technologies that can be used which also influence how traceability is achieved in detail. Therefore the section below is only a rough outline of the different tasks that need to be performed.

Traceability can be achieved by following two steps. The first step is an in depth analysis of the artifacts whereas the second step utilizes the findings of the analysis. All this can be done by using specialized tools or by creating the needed analysis and evaluation capabilities for already existing tools. One of these two options must be applied to create the capabilities to handle the artifacts.

First Step: Analysis

The information contained by the artifacts must be analyzed. The information needs to be evaluated in regard to its relevance for sharing. The correlations between artifacts must be

analyzed too. Correlations are important because they are used to link artifacts together. Theses links are valuable information that can be utilized.

Second Step: Evaluating the Analysis Results
In this step the information gained in the analysis is used. All artifacts containing valuable information must be transformed into a common data format. The transformed artifacts will be linked with each other according to the correlations identified in the analysis part. The links between the artifacts are created by using specialized tools or applying linking capabilities to existing tools.

2.1.3 Advantages of Traceability

Traceability can vastly improve the efficiency of software development and software maintenance. It also greatly influences the workflow within any project it is applied to.

Improving Tasks and Workflow
Traceability streamlines tasks in the production cycle. An example is better control of tasks by tracking the progress which is useful because it directly influences the success of a project by identifying problems in the workflow more easily. This improves the process of decision making. Traceability leads to a reduction in errors created during a task and improves the speed in which errors can be detected and corrected. In addition traceability creates the capability of evaluating if a requirement is satisfied by a software product and can also help to ensure that the implementation of the required features is sufficiently tested.

Enabling Concept
Traceability enables the use of different tasks and techniques. An example is the impact analysis which is part of the change management and improves the quality of a software product. The correlations between different artifacts are made visible by the use of traceability. These correlations can be used by the impact analysis which directly influences the efficiency of the maintenance process of the final software product. The influences of changes to the software product can be identified more easily. Traceability can be used to automate tasks such as the impact analysis which greatly reduces the risks involved in manually handling tasks.

2.1.4 Inhibiting Factors

There is a number of inhibiting factors for traceability which can cause problems and difficulties for the process. I will focus on the three most important ones.

Inhomogeneous or Insufficient Artifact Documentation
Artifact documentations can be in a varied level of detail and can be held in different data formats. This makes analyzing the artifacts and evaluating the results difficult. If the detail

level of documentation is not sufficient it cannot be correctly analyzed. Therefore artifacts may be ignored entirely. This is problematic because it might lead to loss of information relevant for sharing.

Missing Standard

Lacking a standard hinders the analysis of the artifacts and evaluating the findings. A standard helps in identifying which artifacts need to be traced. In addition it helps analyzing complex artifacts as well as improving data integrity and consistency. The decision which technologies will be used to establish traceability capabilities has to be done carefully. This is a challenging task as it requires a high amount of know-how and expertise. Lacking a standard can create the possibility of creating redundancies by holding the same artifact in different tools and storages leaving the created system prone to errors.

Lack of Know-How and expertise

All tasks in the domain of traceability require a certain level of know-how and expertise. If this level is not available the different tasks will create low quality results which will reduce the overall quality of the achieved traceability. A low level of quality can cause problems. Improving the quality is more expensive the later it must be done.

2.1.5 Solutions

Lack of Know-How and expertise

Solving this problem includes hiring experts who have the required abilities. Another option is training workers so that they can acquire the required abilities. This can be achieved by either using experts as teachers or by letting the workers acquire the abilities while working on a project.

Missing Standard

To solve this problem a new standard must be created or an already existing standard must be used. The standard should include two parts. The first part focuses on traceability on a functional level while the second part focuses on a technical level. The higher the standard's quality is the lower the chance to encounter problems while applying it.

Part I: Functional Level

The first part helps in the analysis of the artifacts. It must specify which artifacts should be used in regard to the information contained. The standard should help in identifying important information.

Part II: Technical Level

The second part must specify which technologies should be applied. The technologies must be chosen carefully because they have to satisfy the needs of the traceability process. Choosing the best technologies for the task at hand requires an in depth analysis.

Inhomogeneous or Insufficient Artifact Documentation

There are two basic solutions for this problem. The first option is trying to work with the documentation that is available. The second is a more radical approach used if the quality of the available documentation is insufficient to work with.

First Option: Working with the Available Documentation

The first option is based on using the available documentation. This is possible if the level of quality is high enough. The level of documentation may vary from one artifact to another. This makes the analysis more difficult as the process cannot be standardized across all available artifacts. The process of analysis can be supported by guide lines that specify the workflow on a functional level.

Second Option: Improving the Quality of the Documentation

The second option is to create a new documentation of the needed quality level for the artifacts. The quality level must be specified first so that it matches the requirements of the analysis process. Choosing this option is the last resort choice if working with the available documentation is impossible or if no documentation is available at all. Automated processes can help in creating the documentation.

2.2 Interoperability

2.2.1 Definition

In software engineering the term interoperability describes the ability of a system to interact with another system. This includes the exchange of data and also the common usage of shared data. The size of the systems is not limited and can either describe the interaction of complete software products or just parts or components of a software system. Interoperability itself is not a technology but an overall concept.

Levels of Interoperability

There are two levels of interoperability, the syntactic and the semantic level. The syntactic level focuses on the exchange of data. The semantic level focuses on the interpretation of the exchanged data. If both levels are satisfied it is called full or true interoperability.

First Level: Syntactic Interoperability

Syntactic interoperability describes the ability to exchange data between different systems in a meaningful manner. The data itself must not be altered in the exchange. This ensures that the second level of interoperability can be applied.

Second Level: Semantic Interoperability

Semantic interoperability is only possible to apply if syntactic interoperability was already applied. This ensures that the data is exchanged in a way that allows accurate and correct interpretation. Operating on the data becomes possible.

2.2.2 Advantages of Interoperability

Interoperability has several advantages. The most important ones are the three described below.

Improve Communication

Interoperability increases the quality of cross domain communication. This does include the exchange of information and knowledge of different experts working on a software product. If these experts e.g. a requirement engineer and a software tester can more easily interact with each other it leads to a reduction of problems in the software development process. If the exchange of data between different tools is a complex task the willingness of communication will be reduced.

Improving Workflow Efficiency

Solving problems that are caused by a lack of communication and cross domain interaction of experts can be expensive and time consuming. Identifying the actual problem can be difficult. The lack of efficient communication hinders identifying how a certain part of the system should work or behave. Interoperability helps in software development because it allows sharing domain specific information outside of the domain in which the information was created.

Enabling Concept

Interoperability is an enabling technology for traceability which is a very important aspect in software engineering.

2.2.3 Achieving Interoperability

There is one concept for every level of interoperability. The first concept focuses on the data itself. The second concept focuses on the exchange of data. Usually there is the need to apply different technologies to satisfy the requirements of a concept.

First Concept: Data

The first concept focuses on the data which needs to be in a state where it can be exchanged successfully. This state must be a common data format that can be used in several systems. The transformation process has to take place whenever data is exchanged. The new data format is only used in the exchange.

Every system can hold a representation of the data in a format it can use. The system does not need to be changed to work with new data formats. This is an advantage because it is almost impossible to create one common data format every system can use.

The use of a standardized data format greatly reduces the complexity of achieving successful interoperability. Creating a data format requires know-how and expertise to help in identifying the requirements of the new data format.

Second Concept: Exchange of Data

The second concept focuses on the exchange of data which needs to be specified so it can be used in every system. If the exchange is not specified the exchange can alter the data and therefore the information contained. All operations on the exchanged and altered data can create wrong results. If this is not detected the results are used in other operations. Correcting this will be a complex and expensive task.

The use of standardized communication protocols is mandatory. This creates a common way to exchange data which ensures the data is not manipulated or altered. This is done by establishing common interfaces and common methods.

The interfaces and methods must be stable and public. This means they must not change so that other systems do not have to be altered. If both of these requirements are satisfied the standard can be implemented and used almost without effort in any system.

Creating Interoperability Capabilities

Interoperability can be achieved by creating proprietary adaptors for a specific combination of systems. A better solution is to use a standard that achieves interoperability for any number of systems that are based on it.

Proprietary Solution

The disadvantage of a proprietary solution is that one proprietary adaptor must be created for each combination of systems. The number of proprietary adaptors per system is therefore as high as the number of systems it has to interact with. To create and maintain these adaptors can be a very complex and therefore expensive task. In an environment of many different systems this can even prove to be an impossible task.

Solution based on Standards

The use of a standard reduces the number of adaptors that are created for a single system to one. Other systems can now interact and communicate with the system offering the standardized adaptor solution. This is very cost efficient because the number of adaptors is massively reduced.

2.2.4 Inhibiting Factors

There are also inhibiting factors which strongly influence interoperability. Following are the three most important factors that influence interoperability capabilities.

Sustaining Uniqueness of Products

One of these factors is the need of every system provider to establish its own products and be ahead of other competitors. This is sometimes done by including small changes and alterations in a new revision. The communication of these changes is then delayed to stay ahead of the competition. The competition has to adapt to the new changes which requires time. This practice is used to improve the position of the original vendor and its products.

Patents on Data

It often happens that data and data formats are protected by patents. The problem is that the data cannot be used or generates additional cost.

Errors and Limitations

If a specific product is dominant on the market and used as a basis for other solutions all errors and limitations included in the product are carried over. All systems based on this product will have the problem of having to work around these problems. This will reduce the quality of the solution.

2.2.5 Solutions

Sustaining Uniqueness of Products

If a community is creating a standard it is impossible for a single member of the group to slow down or block the process. If the standard is created and used by several parties it is no longer possible to withhold changes to a system. This is the case because changes and alterations in the software need to always match the current state of the standard. If this is not done it is only a disadvantage of the product not correctly implementing the standard.

Patents on Data

The solution to the problem of data being encumbered by patents can be solved by not building upon this data. Therefore the patented data format must only be used if there is no alternative. The basis for a common data format should not be a patented data format to avoid problems.

Errors and Limitations

The solution is to not to base the new data format on error prone sources.

2.3 Linked Data

2.3.1 Definition

Linked data also called linked open data is a method of publishing information. It is a core concept for the semantic web which can also serve as a concept to achieve interoperability and traceability.

Linked data has two aspects. The first aspect is that the information will be made available as raw data and the second aspect is that data will be connected to other data. The data needs to be prepared according to the first aspect to be able to implement the ideas of the second aspect.

Linked data builds upon technologies like the Hypertext Transfer Protocol (Http), the Unified Resource Identifier (URI) and the Resource Description Framework (RDF).

2.3.2 Advantages of Linked Data

The three most important advantages of linked data are listed below.

Improving Quality and Accessibility of Information

If data is enriched with links to other similar or otherwise relevant data the quality of the original data is increased. It can enable users to find other useful information more easily.

If the information is provided in a basic data format the information can be used more easily because a standardized method to access the uniform data can be used.

Improving Availability of Information

Linked data increases availability of information. The information is available in a raw data format which makes it possible to access information independent from the way it is accessed. Decoupling the data from the representation allows for a wider use of that specific data set. Additionally holding information in a raw format renders the need of holding information in different formats obsolete. This is important because redundancies lead to increased effort in maintenance and create errors.

Enabling Concept

Linked data is an enabling concept. It creates the possibility to create interoperability and traceability capabilities. Both of these are important for the success of any project. Linked data is also a core concept of the semantic web proposed by the World Wide Web Consortium (W3C). The capability to serve as an enabling concept increases the value of using linked data.

2.3.3 Achieving Linked Data

There are three steps that need to be taken to satisfy the requirements of linked data. The first step concerns itself with the data format. The second step involves the identification of the data. The third and last step is about making the data available.

In addition the data must not be protected by any patents or be otherwise inaccessible. This is needed because only freely accessible data can be linked.

First Step

The available information has to be in the form of raw data. This creates the possibility of linking the raw data to other sources of information (raw data). This is an important step

because it supports the basic concept of linked data. This is the idea of being able to link information with other pieces of information to enhance its value.

This is done by using the Resource Description Framework (RDF). RDF can be used as a way to model and describe information. A rough outline of RDF can be found in the section (see 3.2, page 19).

Second Step

If the information is now available as raw data it can be linked to other pieces of information. To efficiently map information there is the need for unique names to identify and locate specific data.

This is done by using the concept of Unified Resource Identifiers (URIs). A URI is a string of characters which will be generated by the use of a set of generation rules. It is used to identify resources. A rough outline of URI can be found in the section (see 3.4, page 21).

Third Step

The last step involves making the data available online. This creates the need for a way to access this data.

The technology used for this step is the Hypertext Transfer Protocol (Http). Http is a protocol used for exchanging data. A rough outline of Http can be found in the section (see 3.5, page 21).

3 Core Technologies

3.1 Representational State Transfer (REST)

3.1.1 Definition

The Representational State Transfer (REST) is an architecture concept developed by the World Wide Web Consortium (W3C). REST is a resource-oriented architecture. The REST architecture uses clients and servers. Clients send requests to servers which are then answered by responses. Requests and responses usually transfer representations of resources.

3.1.2 Advantages of Representational State Transfer (REST)

REST has many advantages. The two most important ones are listed below.

Scalability and Performance

The concepts used in REST make it an architecture that can be scaled very easily. The ability to scale the system up and down greatly influences performance. This makes it possible to adapt the system to the task at hand. This can greatly influence the need for time and money.

Common Interface

REST uses a common interface between clients and servers. This interface is very well structured and easy to understand. The standardization of used methods makes REST an easy to use architecture. This is also the case because of the simplicity of the applied concepts.

3.1.3 Achieving Representational State Transfer (REST)

REST specifies five basic constraints. These are used to specify which aspects a system must have. The components can be implemented freely because REST does not specify how a component has to fulfill the constraints.

Client-Server

Applications are divided into clients and servers which can communicate via a common interface. This allows for clients and servers to be able to evolve independently. This ensures separation of concerns. In addition it allows for low coupling while achieving high cohesion which are important design paradigm.

Layered System

This means that there may not be a direct connection between the client and the server. This is needed to increase performance and scalability. In addition the client must not be able to gather information if it is directly connected to the server or not. The connection must always be in a form where there is no difference for the client how exactly the connection is established.

Stateless

Stateless means that the server must not hold any information regarding the state of the client. Therefore the client has to include all information that is needed by the server in the request. The client and server may hold information regarding their own state.

Cacheable

The client has to be able to cache responses of the server. Therefore every response send by the server needs to be explicitly marked as cacheable. This eliminates the need for some client server communication as the client can reuse older responses of a server.

Uniform Interface

The interface between client and server must be uniform. It has to adhere to the following aspects. It has to be able to identify resources, allow the manipulation and alteration of resources. In addition it has to provide the capability of creating messages that include how the information itself can be used.

3.2 ResourceDescriptionFramework (RDF)

3.2.1 Definition

ResourceDescriptionFramework (RDF) is a metadata model for data. It is a concept for the basic modeling and description of information.

RDF uses a construct called triple to describe information. A triple always consists of a subject, a predicate, also called property, and an object. A set of these triples is called RDF

graph. A RDF triple would translate into two nodes and one edge. The subject and the object would form the nodes. The predicate is the edge that connects the nodes.

A node is either an URI, a literal or blank. Blank is used if the node does not have a form of identification.

3.2.2 Advantages of ResourceDescriptionFramework (RDF)

The most important advantages RDF has are the following three.

Triples

RDF stores all information in the form of triples. This is a very efficient and easy to understand concept. Triples can be easily stored. This is an advantage over many other concepts because it avoids the need of variable length fields. This is not the case with RDF as the triple always needs the same space.

Graph

Every RDF triple or set of triples is a directed graph. This provides RDF with all the advantages of storing information in a graph. It allows the use of all operations that can be used on a graph.

Fact Basis

Every RDF model can be used and interpreted as a fact basis. This allows the inference of new information based on the already stored triples. This makes the process of storing information in RDF even more advantageous as it can provide more information.

3.3 ExtensibleMarkupLanguage (XML)

3.3.1 Definition

The *ExtensibleMarkupLanguage* (XML) is a markup language used to represent data. The XML format is used for textually representing data or data structures. The format used is readable by humans and machines alike.

3.3.2 Advantages of ExtensibleMarkupLanguage

The following is the most important advantage of XML. All other advantages are derived from this one.

Simplicity

XML is a very simple technology which is none the less very powerful. The fact that xml is a very simple concept makes it very easy to apply. This simplicity allows the use of XML in classic applications as well as in internet based ones. The fact that XML is only used to store data and can be used by any application makes it a very stable format. This is important because using older formats may be a limiting factor for an application. All information can

be structured with XML. The ability to represent any form of information renders the need for different data formats and transformation of different data formats obsolete.XML allows the ability for humans and machines to read the file. This increases efficiency because there is no need for two separate data formats.

3.4 UniformResourceIdentifier (URI)

3.4.1 Definition

The *UniformResourceIdentifier* (URI) is used for two things. The more obvious one is identification of resources. The second one is allowing interaction in a network. Interaction across several systems creates the need to identify resource in a unique fashion to correctly address them.

3.4.2 Advantages of UniformResourceIdentifier (URI)

The biggest advantage of URIs is that it is a standard that has been used for a long time. This makes it a very well understood standard which is used in many different systems. The longevity of the URI standard is proof of its quality.

3.5 HypertextTransferProtocol (Http)

3.5.1 Definition

The *HypertextTransferProtocol* (Http) is a protocol used for exchanging data. It is a protocol used for sending and receiving hypertext that is used in the context of the client-server model. To achieve this it is based on requests and responses. Http is a protocol that is based on sending all valid information regarding the state of the sending component within the request. This is useful because it allows the use of concept that demand that the server does not keep track of client states i.e. REST.

3.5.2 Advantages of HypertextTransferProtocol (Http)

Complete Information in Requests and Responses

Requests and responses including complete information allow the use of Http in many systems. This is the case because there is no need for additional technologies to enable the server to keep track of client states. This makes Http a very good technology for simple communication.

Simplicity

Http is a very basic protocol which makes it widely usable. Additionally it is easy to understand and apply. This makes choosing a communication protocol easy because of the simplicity of Http while being a powerful protocol.

4 Open Services for Lifecycle Collaboration (OSLC)

4.1 Definition

The Open Services for Lifecycle Collaboration (OSLC) is best described in the following quotation: "Open Services for Lifecycle Collaboration (OSLC) is a community of software developers, operations experts, and organizations that are working to **standardize the way that software lifecycle tools can share data** with one another." [highlighted in original][1]

OSLC has several members that develop software using OSLC, work on the specifications and participate in workgroups. Among them are well known companies like EADS, IBM, Ericsson, General Motors, Oracle, Shell, Siemens and Boeing to name just a few.

The standards and specifications created by OSLC can be used to achieve tool interoperability and create traceability capabilities.

4.2 Principles and Philosophy

OSLC stands on three pillars (goals, process, specification).[2] Each pillar has aspects to it that influence the community, the specifications and standards as well as the process of creating them.

---

[1] OSLC 2013-1

[2] OSLC 2013-2

Pillar 1: Goals

Principles and Philosophy	Description
Improve Connections	Another goal is to make the communication between different lifecycle tools easier by creating flexible and robust connections. This means that the connections are not point-to-point connections but connect the tool to a common cloud of other tools that use OSLC.
Improve Data Usage	The final goal of OSLC is to make more data available. This allows more complex operations on the data as well as improving the functionality of a given tool.
Improve Process of Integration	One goal of OSLC is to make the integration of life tools easier. To achieve this they have created a new standard.

Pillar 2: Process

Principles and Philosophy	Description
Discuss Problems and Solutions	OSLC is heavily based on discussing important topics in special workgroups. This is done to create the minimal amount of information, scenarios and use cases that are needed for the successful support of common tasks and interactions in the process of integrating tools.
Gather Experts	OSLC brings together experts of the different domains like quality, requirements and architecture management to name just a few. This improves the quality of the standard and generates additional know-how and expertise.
Spreading Open Specifications	OSLC focuses on spreading the open standards that are created so that they can be used in a wide field of applications and tools. This is done by the agreement of all members to not have patents on ideas and implementations in the scope of OSLC.

Pillar 3: Specification

Principles and Philosophy	Description
Evolving	OSLC standards and specifications are involving over time to be able to address more use cases and scenarios. This is done so that the standards and specifications can always match the current requirements of the software engineering processes.
Implemented and Tested	There are several example implementations including tests to make sure that the OSLC standards and specifications are sufficient to be effectively and efficiently used in the context of applications and systems.
Inspired by Web Technologies	OSLC specifications and standards build on well-known and widely used web and internet standards. This includes the concepts specified by linked data as well as the use of Hypertext Transfer Protocol (HTTP), Representational State Transfer (REST) and Uniform Resource Identifier (URI).
Minimalistic	All specifications defined by OSLC are minimalistic therefore specifications are only as complex as needed for common scenarios. OSLC standards and specifications are not created to address every possible use case to keep the complexity on a low level.

4.3 Choosing OSLC

In this section I will list the advantages of OSLC that lead to choosing it as the core source of specifications for this thesis.

- Huge Community
  The OSLC community is a very big one including well-known companies that work on creating, improving and using the specifications and standards. This makes OSLC relevant because the more companies use a specification and are interested to work on it, the higher the chance that the specifications will always be up to par with the rest of the competition and therefore will be long lasting and available.
- New Specification
  OSLC specifications and standards are rather new and therefore really interesting to work with. In addition it is still relevant to have a proof of concept which is not necessary for well-established specifications and technologies.
- Fast Growing
  OSLC is a very fast growing community which makes it very interesting because this thesis can be part of this process.

4.4 Structure

OSLC is structuring its content by dividing it in two areas. These are the core and the domain groups. The core specifies constructs and concepts that need to be generally used and applied. The domain groups include additional constructs and concepts relevant for the implementation of OSLC in a specific domain.

The core includes the general resources, properties and services that are not domain specific. For more information look into the core specification: http://open-services.net/bin/view/Main/OslcCoreSpecification

The domain groups focus on creating domain specific standards content. This includes resources and properties that are only useful in the context of the domain and are therefore not include in the core. The domains that are currently supported by OSLC include:

- Architecture Management
- Asset Management
- Automation
- Change Management
- Quality Management
- Reconciliation
- Requirements Management

4.5 OSLC QM

OSLC QM includes the following five domain specific resources:

Resource	Description[3]
TestCase	"Defines the criteria which determine whether a system exhibits the correct behavior under a specific set of circumstances."
TestExecutionRecord	"Planning for execution of a test."
TestPlan	"Defines the overall process and strategy for testing a system."
TestScript	"Defines a program or list of steps used to conduct a test."
TestResult	"Describes the outcome of attempting to execute a test."

For more information about the Quality Management domain group visit: http://open-services.net/bin/view/Main/QmSpecificationV2

[3] McMahan 2011

4.6 Concepts

This is a rough outline of the OSLC constructs to give a simple overview. Further information as well as specific details can be found in the specifications.[4] OSLC is based on two basic principles. These two are used to separate data from operations.

The first is that artifacts are mapped to equivalent OSLC resources. The information of the artifacts is represented by OSLC properties which are described in the resource shape. Every OSLC resource has a corresponding resource shape that describes it.

The second is that these resources can be operated on by the use of OSLC services. These services include creation factories and query capabilities.

4.7 Technologies

As stated in the “Principles and Philosophy” section (see 4.2, page 23) OSLC builds upon the web technologies such as Hypertext Transfer Protocol (HTTP), Representational State Transfer (REST) and Uniform Resource Identifier (URI). Additionally it also uses Resource Description Framework (RDF) and Extensible Markup Language (XML). These technologies form a powerful set of concepts which are sufficient for solving any task.

4.8 Lyo

Lyo is a Software Development Kit (SDK) for OSLC. It provides basic concepts and constructs to support the implementation of the OSLC specification in the context of adaptors used in tools. More information can be found in the Eclipse Lyo Wiki.[5]

---

[4] McMahan 2011; Arwe 2013

[5] Fiedler 2013-1

5 Related Work

5.1 ModelBus

5.1.1 Definition

ModelBus is a tool integration framework used to build seamlessly integrated tool environments in the domain of system engineering and focusing on application lifecycle management. ModelBus is used to achieve interoperability between tools and can therefore create traceability capabilities.

5.1.2 Principles and Philosophy

ModelBus is best described by the following six distinct key features.[6]

Principles and Philosophy	Description
Automation	ModelBus allows increasing the efficiency of the workflows by using automation. The automation can either be started by user commands or other means of triggering.
Built on Industry Standards	ModelBus relies heavily on industry standards which include: Web Services (Axis2), BPMN, BPEL, OMG standards OCL, UML, MOF (EMF) as well as JMS.
Distributed and Heterogeneous	ModelBus allows heterogeneous tools to be integrated. Additionally it is a distributed system because all tool adaptors transform data into the needed formats which can be used for communication by ModelBus.

[6] Ritter

Principles and Philosophy	Description
Extensible and Customizable	ModelBus can be applied in every development environment because it is customizable to the extent that it can fit many possible setups. The number of tools that support ModelBus can be extended by using the ModelBus software to create new adaptors.
Support of Large and Complex Models	ModelBus uses a repository for version control, partial checkouts as well as merging and branching of model fragments and models.
Transparent Model Management	ModelBus increases the transparency of the development process by allowing easy identification and access of models.

5.1.3 Concepts

ModelBus is used to take artifacts created by different tools and put them into the Model Repository. This is done by implementing an adaptor for the tool and then connecting to the repository via the bus. ModelBus also includes a repository browser to use the repository.

ModelBus also includes a Service Repository that includes different services that are already included in ModelBus or are created by implementing them. These services are then used to wrap functionality of a tool so that it can be used outside of the tool if the using party is also connected to the bus.

ModelBus goal is to create interoperability of the first and second level. This is done by creating and then using transformation services that transport information from one data format into another so that the data can be used in different tools. There is no common data format therefore every transformation is a one to one transformation for the combination of two tools.

ModelBus uses traceability data that is added in the repository to realize traceability. The added data includes basic constructs like timestamps for creation and modification as well as information about the creator of a specific resource.

5.1.4 Comparison

ModelBus and OSLC have different goals and also different philosophies. Comparing these two is therefore not useful because both goals and philosophies are good for their purpose but cannot be applied to the goals achieved by the other technology.

6 Analysis

6.1 Overview

In this chapter I summarize the findings of my analysis for the "Rational Rhapsody Test Conductor" constructs as well as my criteria for evaluating them for mapping. I have identified constructs as either relevant or irrelevant. These and the criteria I used are mentioned in the specific sections.

6.2 Relevant Constructs

6.2.1 Criteria for Relevant Constructs

Only constructs holding valuable and heavily used information should be included in the mapping process because if there are too many resources the amount of time needed to find the relevant information grows.

Listed below are the criteria a construct must satisfy for being classified as relevant for mapping. All the criteria below are aspects that make a construct relevant in regard to traceability and therefore the criteria below are more detailed and specific.

- Management and Control
  This criterion is applied if the construct is relevant from a management or controlling perspective and should therefore be made available.
- Cross Domain
  This criterion is applied if the construct includes information that is relevant outside of the testing domain and should be accessible outside of the tool it is created and used in.

- Crucial Information
  This criterion is applied if the construct includes information that is crucial and therefore must be made available outside of the tool it is create and used in. This is a different criterion from "cross domain" because it is crucial information inside its own domain.

Below is the list of all "Rational Rhapsody Test Conductor" constructs that I identified as relevant for traceability purposes using the above-mentioned criteria. The detailed analysis of every construct and reason for mapping it can be found below the following table.

Each table entry includes the name of the "Rational Rhapsody Test Conductor" construct, a short description and the criteria used for identifying.

Construct	Description	Criteria		
		Manage-ment and control	Cross domain	Crucial information
CoverageResult	Result specifies which model elements are tested by which test cases	✓		✓
Scheduler	Stereotype for test components that control the execution of test cases	✓		✓
System under Test (SUT)	Part of a system that is tested by a test case	✓	✓	✓
TestCase	Specifies the correct behavior of the SUT, input and expected output	✓	✓	✓
TestObjective	Links test cases to requirements	✓		✓
TestRealTimeResult	Result of the test case execution with tool "TestRealTime"	✓		✓
TestResult	Final result of the test case execution, pass or fail of test case	✓		✓
TestScenario	Stereotype for diagrams, diagrams specify specific steps in test case execution	✓		✓

6.2.2 CoverageResult

The CoverageResult documents which elements of the model are covered by one or many test cases.

Advice: The CoverageResult should be made available by mapping it to an OSLC resource.

Reason: The CoverageResult is important from a management and control perspective because it allows controlling if the system is sufficiently tested. This information is needed to take further steps if there are parts of the system that are not covered by tests.

The CoverageResult is important to decide if the system is sufficiently and correctly tested or if there is the need to specify further tests.

6.2.3 Scheduler

The scheduler is used to control the execution of test cases. This includes the activation and termination of different test cases. This helps in controlling and balancing work load on hardware and makes rerunning tests easier.

Advice: The scheduler should be made available by mapping it to an OSLC resource.

Reason: The scheduler is important from a management and control perspective because it allows planning the execution of test cases. This is important because it allows efficiently rerunning test cases for regression test as well as performance balancing while testing.

The scheduler is needed to schedule the execution of test cases.

6.2.4 System under Test (SUT)

The system under test (SUT) is the part of the system that is being tested. The SUT is closely linked to the test case and the requirement it satisfies. The complexity of the SUT depends on the functionality that is being tested. To satisfy the needs of different levels of testing the SUT can be composed of different parts of the system.

Advice: The SUT should be made available by mapping it to an OSLC resource.

Reason: The SUT is important from a management and control perspective because it allows checking if the SUT is correctly linked to requirements and test cases which allows covering all requirements as well as test coverage of the implemented features.

If the SUT is available as an OSLC resource it can be linked to requirements and test cases. These links can either be traced starting from the SUT or from the other direction which increases efficiency and effectively reduces errors from lack of interconnection.

The SUT itself is a crucial piece of information in the domain of testing. It is used to specify which part of a software system is used by a test case.

6.2.5 TestCase

Test cases are constructs included in the test context. A test case specifies which inputs are used to stimulate the SUT and which the correct and expected results are. The TestCase is

specifying the behavior of a test and also specifies how the test components are interacting with the SUT to realize the test objective.

Advice: The TestCase should be made available by mapping it to an OSLC resource.

Reason: The TestCase is important from a management and controlling perspective because it allows correctly covering the system by testing the implementation of the features specified by the different requirements.

The TestCase is also important from a cross domain perspective just like the SUT because it allows correctly linking test cases with requirements and SUTs.

The TestCase is the central point of the testing process as it specifies how the SUT needs to be stimulated and what the expected and correct output and behavior should be.

6.2.6 TestObjective

The TestObjective is used to specify the different objectives of a test case. A test case can have several test objectives that are used to specify the purpose for designing and executing the specific test case. To achieve this, the test objective links the test case to the requirement.

Advice: The information (the link) included in the test objective needs to be made available. This is not done by mapping the test objective to a specific OSLC resource but by including the link in the linked resources.

Reason: The TestObjective is important from a management and control perspective because it specifies why test cases are designed and executed. Being able to track this information is important because it allows reducing costs by avoiding redundant tests. Even more important is to ensure that the system and requirements are correctly covered by tests.

The TestObjective holds crucial information inside of the test domain because it provides the reasons for testing a specific part of the system which allows a controlled and efficient testing process.

6.2.7 TestRealTimeResult

The TestRealTimeResult is the result of the execution of a test case or a test context which can include any number of test cases. This is only available if the execution is done by using the "TestRealTime" tool.

Advice: The TestRealTimeResult should be made available by mapping it to an OSLC resource.

Reason: The TestRealTimeResult is important from a management and control perspective because it allows controlling if the test had the desired outcome or not. This information is needed to take further steps if errors or problems are detected.

The TestRealTimeResult is important because without execution results there is no need for testing. The results are used to check if the testing process is proceeding as planned.

6.2.8 TestResult

The TestResult represents the overall verdict for a single test case and can be used to group test cases together for analysis. The TestResult can have one of the two following values:

- Pass: This value shows that the system under test (SUT) showed the correct behavior for the test case which includes this verdict.
- Fail: This value indicates that the SUT did not show the correct behavior for the test case which includes this verdict.

Advice: The TestResult should be made available by mapping it to an OSLC resource.

Reason: The TestResult is important from a management and control perspective because it allows checking if a test case was executed successfully. It is also used to ensure that the testing process is following the desired schedule.

The TestResult is important because it allows a rough division of test cases to speed up the process of identifying the need for taking further action.

6.2.9 TestScenario

The TestScenario specifies specific steps that are taken in running a test case. This includes sending and evaluating messages as well as general behavior control.

Advice: The TestScenario should be made available by mapping it to an OSLC resource.

Reason: The TestScenario is important from a management and control perspective because it allows controlling the steps that are taken to run a test case. This information is crucial because it allows controlling the correct execution of the test case.

The TestScenario is important because without it there is no way to decide if the test case is correctly used in testing and if the actual test can correctly validate the test case.

6.3 Irrelevant Constructs

6.3.1 Criteria for Irrelevant Constructs

Excluding constructs from the mapping is necessary because making available information that is not needed is expensive in time and money. The resources holding the irrelevant information have to be created and maintained. This causes overhead and has to be avoided.

Listed below are the criteria a construct must satisfy for being classified as irrelevant for mapping.

- Technical Construct (Marks, Tags and Links)
  This criterion is applied if the construct serves a technical purpose without holding relevant information. These constructs are important for the tool to actually work and provide the needed functionality but cannot serve a purpose outside the tool

environment. This is applied to constructs that are used to mark or tag other constructs which includes dependencies between two other constructs. The link between the constructs itself is important but it is not created in mapping the technical construct but creating a relational property for the specific resources that are linked.

- Non-Crucial Information
  This criterion is applied if the construct does not include information that is relevant or important from a traceability perspective which means it is not satisfying any of the criteria for relevant constructs. This criterion is applied to many constructs that only provide a limited amount of information which is not sufficient to be mapped. Additionally it is also applied if the information stored in this construct is specified in another construct. Therefore the information is non-crucial because it would create a redundant source for the same information.
- Grouping Mechanism
  This criterion is applied if the construct is a grouping mechanism and therefore holds important information indirectly. Therefore the construct should not be mapped but the included constructs that are grouped together by the mechanism.

Below is the list of all the Rational "Rhapsody Test Conductor" constructs that I identified as irrelevant for traceability purposes using the above mentioned criteria. This section does not include the detailed analysis of all the constructs but the core results can be found in the following table.

Each table entry includes the name of the "Rational Rhapsody Test Conductor" construct, a short description and the criteria used for identifying.

Construct	Description	Criteria		
		Technical construct	Non-crucial information	Grouping mechanism
Arbiter	Evaluation of test results, sets verdict for test case or test context (all test cases in the context)	✓	✓	
ArbiterInstance	Instance of arbiter test components which control execution in testing mode based on assertions	✓	✓	
ControlArbiter	Dependency between test component and arbiter	✓	✓	
CppUnitConfig	Provides properties for custom CppUnit testing integration	✓	✓	

Construct	Description	Criteria		
		Technical construct	Non-crucial information	Grouping mechanism
CppUnitContext	Provides tags for CppUnit testing integration	✓	✓	
CUnitConfig	Provide tags for custom CUnit integration	✓	✓	
CUnitContext	Provide properties for CUnit testing integration	✓	✓	
DefaultOperation	Default behavior if test case does not specify differing behavior	✓	✓	
DriverOperation	Generates automatically for test components, used in SUT stimulation	✓	✓	
Instantiated	Tag for associations that are instantiated with valid links	✓	✓	
NoConsoleApp	Suppresses opening of app's console	✓	✓	
ParameterTable	Marks file as parameter table which includes test parameter values	✓	✓	
Replacement	Used to mark replacing test components	✓	✓	
RTC_InstInfo	Enables use of tags in test execution, includes monitoring messages and default behavior	✓	✓	
RTC_MsgInfo	Enables use of tags in driver operations	✓	✓	
RTC_RefInfo	Used for internal message identification	✓	✓	
Scheduled	Dependency between test context and scheduler test component	✓	✓	
Scheduler	Controls execution of test components, informs arbiter to generate verdict	✓	✓	
SCTCInstance	Mark for instances of test components of state chart test cases	✓	✓	
SDInstance	Represents instance of TestScenario	✓	✓	
StatechartTestCase	Dependency between state chart and test component	✓	✓	
Stubbed	Mark for stubbed operations, meaning operations was altered for testing purposes	✓	✓	

Construct	Description	Criteria		
		Technical construct	Non-crucial information	Grouping mechanism
StubbedOperation	Operation for which a test case specified non default behavior	✓	✓	
StubOperation	The specific behavior for a stubbed operation	✓	✓	
TestAction	Action block for test scenarios, block on life line can specify action like asserts to check outputs or load code for complex tasks	✓	✓	
TestActor	New term replacing actors for testing purposes		✓	
TestComponent	Realizes behavior of test case, communicates with SUT and other test components, evaluates if output is correct for given input	✓	✓	
TestComponentInstance	Instance of test components	✓	✓	
TestConfiguration	Collection of connections to SUT and test components, specifies setup of test components during test execution, defines how SUT and test components act in a test context	✓	✓	
TestContext	Includes SUTs and test components and specifies interconnections		✓	✓
TestContextDiagram	Graphic representation of TestContext, does not provide more information than the test context itself		✓	
TestPackage	Container for testing related elements, separates testing from design elements		✓	✓
TestParameter	Marks attribute to be used as parameter	✓	✓	
TestRealTime	Used when using tool "Rational TestRealTime"	✓	✓	
TestRealTimeFile	Used to store results of tool "Rational TestRealTime" executions in model	✓	✓	
TestRequirementMatrix	List of test cases and the associated requirements		✓	✓

Construct	Description	Criteria		
		Technical construct	Non-crucial information	Grouping mechanism
TestResultTable	List of test cases and the associated test results		✓	✓
Unrealized	Tag for unrealized messages, used for filtering	✓	✓	
Use_ParameterTable	Dependency of parameter table and test configuration	✓	✓	
Use_Replacement	Dependency for replacing test components	✓	✓	

6.4 Mapping

This table includes the relevant constructs and the mapping to the corresponding OSLC resource. Following the table is a more detailed description if the mapping process of the construct is not straight forward.

RTC Construct	OSLC Resource
CoverageResult	TestResult (OSLC QM)
Scheduler	TestExecutionRecord (OSLC QM)
SUT	Resource (OSLC AM) The SUT is not an OSLC QM but an Architecture Management Resource and must be included as a property in the OSLC QM test case. One option would be to include a usesSUT property.
TestCase	TestCase (OSLC QM)
TestObjective	TestCase (OSLC QM); Property: validatesRequirement
TestRealTimeResult	TestResult (OSLC QM)
TestResult	TestResult (OSLC QM)
TestScenario	TestScript (OSLC QM)

6.4.1 SUT

The SUT is mapped to an OSLC Architecture Management (OSLC AM) resource. The prototype adaptor included in the scope of this work is an OSLC Quality Management (OSLC QM) adaptor. Therefore the actual mapping of the SUT will not take place in the adaptor.

The SUT is implemented in another tool and will be mapped to the appropriate OSLC resource’s relationship property which will make it available for tracing.

6.4.2 TestObjective

The TestObjective links a test case to a requirement. The TestObjective does not include additional information that should be made available and therefore the test objective is transformed into the OSLC Quality Management (OSLC QM) test case property *validatesRequirement* to contain the link information.

6.4.3 OSLC QM resources without Mapping

The OSLC QM test plan does not have a mapping to a "Rational Rhapsody Test Conductor"(RTC) construct. This is the case because the RTC does not support the test management process and does therefore not include test management related constructs.

7 Scenario

7.1 Pre-Condition

1. Rational Rhapsody Test Conductor is currently running.
2. The prototype adaptor is currently running.
3. The prototype provider is currently running.
4. A Test Conductor project is opened.

7.2 Post-Condition

1. Every project test case is represented by an OSLC QM test case.
2. Every OSLC QM test case is present on the provider.
3. Every OSLC QM test case on the provider is accessible by OSLC consumers.
4. Every OSLC QM test case on the provider is accessible via browser.

7.3 Steps

1. The adaptor reads in the test cases of the project.
2. The adaptor creates OSLC QM test cases based on the retrieved test cases.
3. The adaptor posts the OSLC QM test cases to the provider.
4. The provider makes these OSLC QM test cases accessible by OSLC consumers.
5. The provider makes these OSLC QM test cases accessible by a browser.

8 Requirements

These are the requirements for the prototype implementation that will serve as a proof of concept.

8.1 Traceability

The different constructs of the "Rational Rhapsody Test Conductor" (RTC) needed in the scenario are mapped to the corresponding OSLC QM resources and provided for other users. The provided OSLC QM resources can be accessed by other systems without the need that the RTC is up and running.

8.2 Interoperability

The provider must be able to interact with the adaptors of other tools. Interaction means that other adaptors must be able to access and retrieve OSLC QM resources held by this provider.

The adaptor must be able to interact with the providers associated with other tools. Interaction means this adaptor must be able to access and retrieve OSLC resources held by other providers.

8.3 Specification

The implementation must reflect the state of the OSLC specification. The version of the core specification is "OSLC Core Specification Version 2.0"[7] and the version of the QM specification "OSLC QM Specification Version 2.0"[8]. The state must therefore reflect the

[7] Arwe 2013
[8] Mahan 2011

specification in the above mentioned versions. This includes the setup of the different OSLC QM resources and properties as well as the described services.

8.4 Simplicity

The implementation must be kept as simple as possible. This means it must only support the minimum of the functionality needed for the scenario. This will make sure the proof of concept will be easily understood in its basic functionality.

8.5 Modularity and Extensibility

The implementation must be kept modular. This will ensure that the prototype's components have low coupling while achieving high cohesion.

The prototype must be implemented in a way to allow extensions of functionality aspects. This is necessary to create a full business solution from the basis of the prototype. Furthermore it increases the value of the prototype as it can be used for further implementation instead of being a proof of concept.

9 Design

In this part I describe the design for the OSLC QM adaptor prototype for "Rational Rhapsody Test Conductor" (RTC) that I created as a proof of concept. The prototype will only include the functionality needed to be successfully used in the described scenario.

Following is an abstract overview of the OSLC adaptor and its functionality. Every tool has its own OSLC adaptor in addition to its OSLC provider (see Fig. 5, page 43). A tool's OSLC adaptor is able to communicate with OSLC providers. The OSLC adaptor can transform constructs included in the tool to OSLC resources. These resources are then sent to the provider. It can consume OSLC resources of an OSLC provider (see Fig. 6, page 44). The OSLC adaptor can also consume OSLC resources of the providers of other tools (see Fig. 7, page 44).

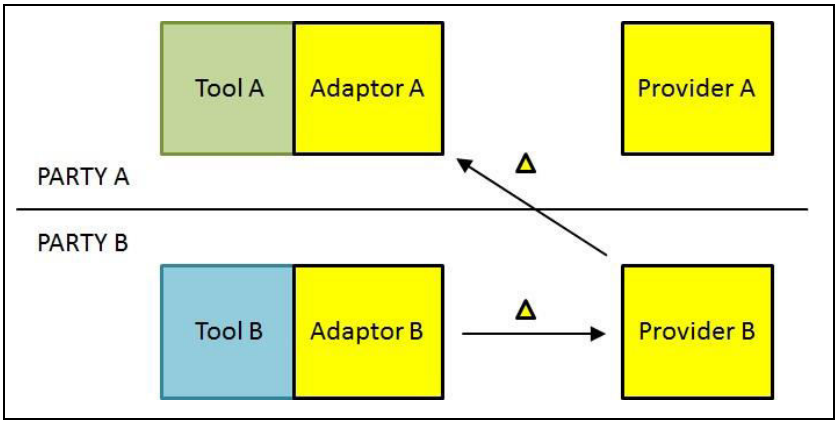


Fig. 5: Tools with implemented OSLC adaptor solution.

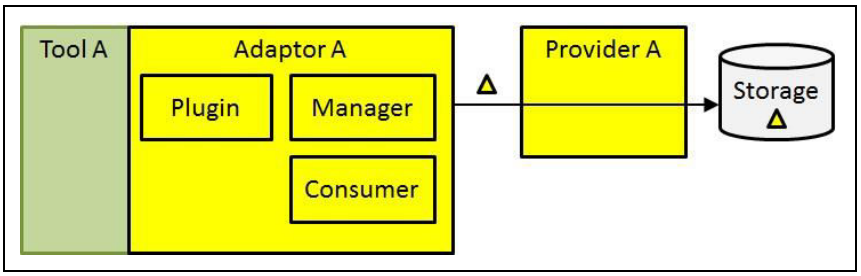


Fig. 6: Principle of the OSLC adaptor uploading OSLC resources to its own provider.

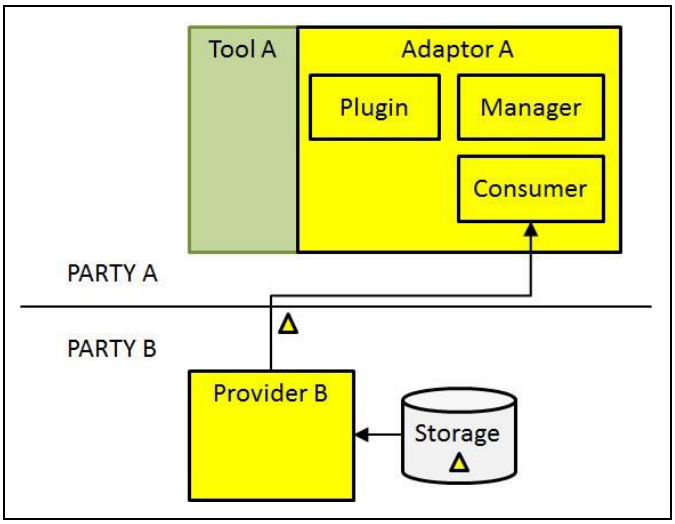


Fig. 7: Principle of OSLC adaptor consuming OSLC resources of a provider.

9.1 Overview

The OSLC QM adaptor prototype will include three parts which are needed to provide the functionality of the adaptor. These three parts are the plugin, the client and the server. Following is a short and abstract description of these parts.

9.1.1 Plugin

The *Plugin* is used to read constructs from "Rational Rhapsody Test Conductor" projects and form OSLC QM resources based on these constructs. The OSLC QM resources will be passed to the client.

9.1.2 Client (Consumer and Manager)

The client transmits the received OSLC QM resources to the server. In addition the client is also used to request OSLC resources from other adaptors. This is done by communicating with the server holding the OSLC resources of the other tool.
In the context of OSLC this client is called consumer.

9.1.3 Server (Provider)

The server receives OSLC QM resources from the client included in the adaptor and makes the resources accessible from outside the tool. In addition the server receives requests from other clients that send the OSLC QM resources for processing purposes. In the context of OSLC this client is called provider.

9.2 Project Structure

Based on the rough outline given in the overview section (see 9.1, page 44) the overall structure of the implementation is divided into three projects. The first includes the adaptor itself. The second includes the provider (server). The third includes basics that are used in both of the other two projects. These projects form the prototype that is created for the scenario in the scenario section (see 7, page 40).

9.2.1 rtc_oslc_adaptor

This project includes the plugin and the client to provide the functionality of the adaptor for the prototype. The client is composed of the consumer and the manager (see Fig. 9, page 48).

This is a separate project because I want to be able to run the adaptor implementation separately from the provider. This allows using different implementations of the provider as well as creating the possibility to use different approaches regarding provider availability. The two most essential approaches are the following. The first is that every adaptor has its own provider while the second is that one provider is used for several tools. The second option is the better choice for a company because it lowers the amount of different providers to one which will make maintenance easier.

9.2.2 rtc_oslc_provider

This project includes the provider to provide the functionality of the server for the prototype to make the resources created in the "*rtc_oslc_adaptor*"-project available outside of the adaptor and therefore outside of the tool (see Fig. 17, page 53).

This is a separate project because I want to create the possibility to easily exchange the provider implementation with a different one. In addition I want to be able to run the provider independently of the adaptor.

In some implementations the provider is an integral part of the adaptor which means it cannot be easily exchanged with a different implementation and that it is only available if the adaptor is running.

9.2.3 rtc_oslc_common

This project includes the basic implementation of the OSLC which are the *TestPlan*, *TestCase*, *TestScript*, *TestExecutionRecord* and the *TestResult*. In addition it includes a number of constants that are needed in the implementation of the prototype (see Fig. 24, page 59).

This is a separate project because the resources and constants are needed in the "*rtc_oslc_adaptor"* and "*rtc_oslc_provider"*-project. If this is not done it will duplicate the code which will lead to a higher amount of time needed to maintain and change parts of the implementation as well as increasing the likelihood of errors.

9.3 System Architecture

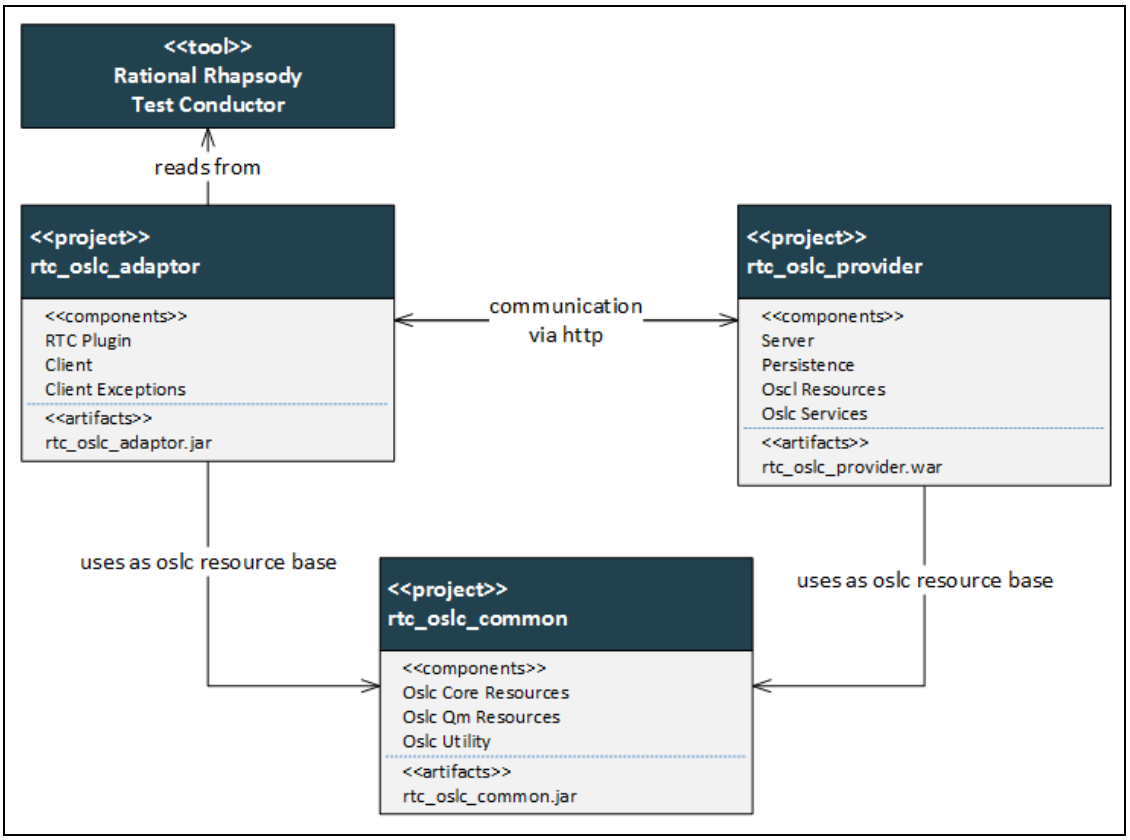


Fig. 8: UML component diagram of the system architecture.

9.4 Project rtc_oslc_adaptor

This project includes the client with the two interfaces *IResourceManager and IResourceConsumer* and the implementing classes *Manager* and *Consumer*. It also includes the Exceptions for the client which are the *NoMatchingQueryCapabilityException*, *NoMatchingCreationFactoryException* and *NoMatchingServiceProviderException* class. The project finally includes the *Plugin* class that is used to interact with the "Rational Rhapsody Test Conductor" (RTC).

9.4.1 Components

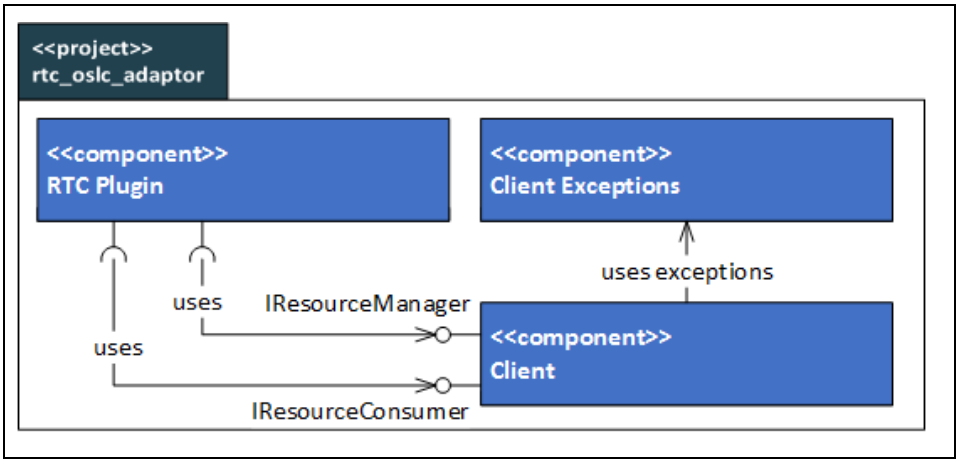


Fig. 9: UML component diagram "rtc_oslc_adaptor"-project (external view).

9.4.2 Client Component

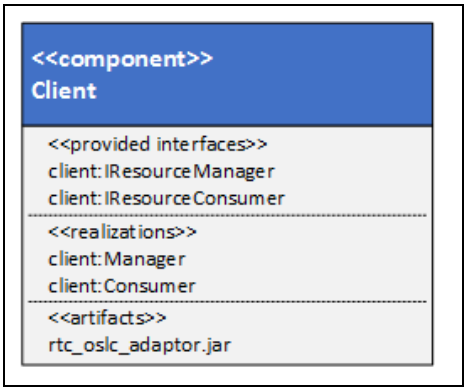


Fig. 10: UML component diagram Client component (internal view).

This is included in a separate component because it includes the functionality of the client. This component is represented by the "*client*"-package. It includes the *IResourceConsumer* and *IResourceManager* interfaces and the *Consumer* and *Manager* classes (see Fig. 10, page 48).

The *IResourceConsumer* is used to specify the interface for adaptor-server communication. The interface includes methods for retrieving resources from the provider as a bulk operation *retrieveResources* or as a single resource *retrieveResource*.

The *Consumer* class is implementing the *IResourceConsumer* interface and provides the functionality for the two specified methods. In addition it includes a method to retrieve a creation-URI from a service provider catalog *getCreationURI.*

The *IResourceManager* is used to specify the interface for adaptor-server communication. The interface includes methods for adding *addResource*, updating *updateResource* and deleting *deleteResource* resources of the provider.

The *Manager* class is implementing the *IResourceManager* interface and provides the functionality for the three specified methods. In addition it includes a method to retrieve a query-URI from a service provider catalog *getCreationURI.*

IResourceConsumer and IResourceManager are used to specify the interface for the client component. The IResourceConsumer includes the methods for resource retrieval which can be used to retrieve a single resource or more. The IResourceManager includes the methods for adding resources to a provider, updating a resource that is already present on a provider and deleting a resource that is held by a provider.

9.4.3 Client Exceptions Component

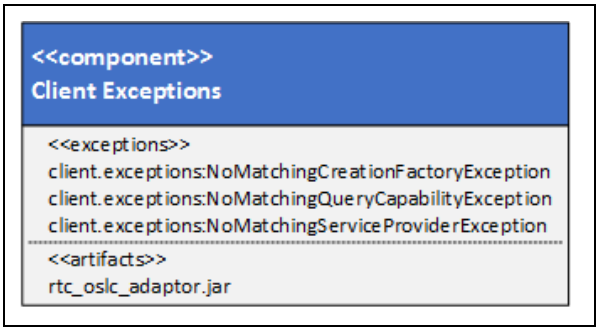


Fig. 11: UML component diagram Client Exceptions component (internal view).

This is included in a separate component because it includes the exceptions for the client. This component is represented by the "*client.exceptions*"-package. It includes the *NoMatchingQueryCapabilityException* (see Fig. 16, page 52), *NoMatchingCreationFactoryException* (see Fig. 15, page 52) and *NoMatchingServiceProviderException* (see Fig. 14, page 52) classes (see Fig. 11, page 49).

The NoMatchingServiceProviderException is thrown by the *getCreationURI*, *getQueryBaseURI*, *addResource*, *retrieveResource* and *retrieveResources* methods. It states that there was no matching service provider for the given resource type included in the given service provider catalog.

The *NoMatchingCreationFactoryException* is thrown by the *getCreationURI* and *addResource* methods. It states that there was no creation factory available for the given resource type included in the given service provider catalog.

The *NoMatchingQueryCapabilityException* is thrown by the *getQueryBaseURI, retrieveResource* and *retrieveResources* methods. It states that there was no query capability available for the given resource type included in the given service provider catalog.

9.4.4 RTC Plugin Component

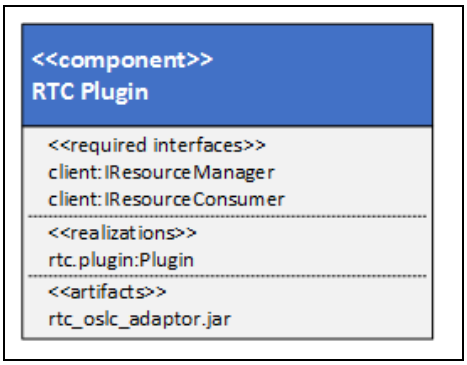


Fig. 12: UML component diagram RTCPlugin component (internal view).

This is included in its own component because it includes the functionality of the "Rational Rhapsody Test Conductor" (RTC) plug-in. This component is represented by the "*rtc.plugin*"-package. It includes the *Plugin* class (see Fig. 12, page 50).

The *Plugin* class inherits methods from the *RPUserPlugin* superclass. The methods are *RhpPluginInit*, *RhpPluginInvokeItem*, *OnMenuItemSelect*, *onTrigger*, *RhpPluginCleanup* and *RhpPluginFinalCleanup*. These methods are called by the RTC to handle the plug-in. The *RhpPluginInit* method includes the functionality to initially read in and create the constructs from the currently opened project in the RTC. The *onTrigger* method includes the functionality to add, update and delete the OSLC QM resources on the provider. This is done on the "After project save" trigger event. The *Plugin* class also includes fields of the type *IResourceConsumer* and *IResourceManager* for the consumer and manager to use the included communication functionality.

9.4.5 Class Diagrams

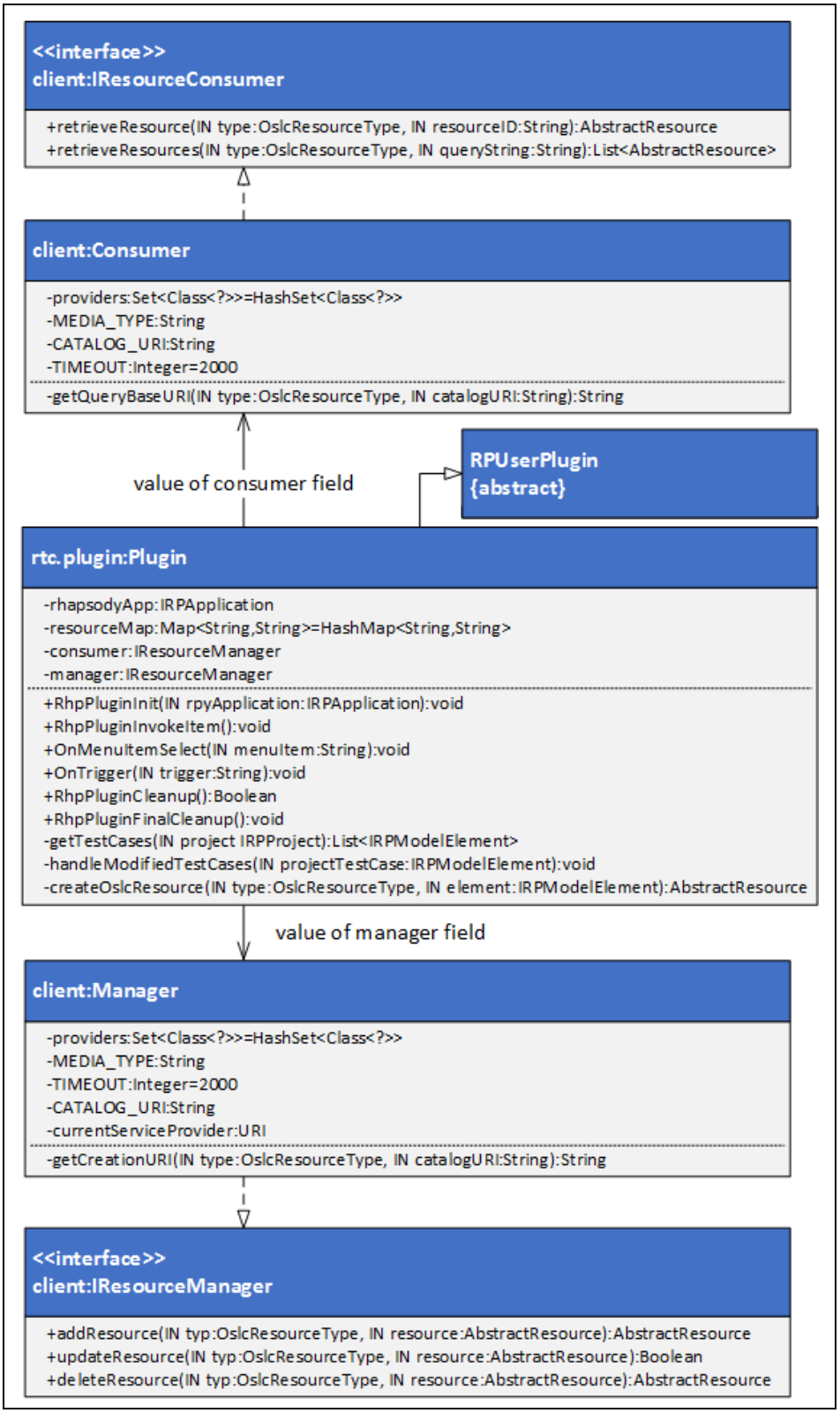


Fig. 13: UML class diagram Plugin, Manager, Consumer classes

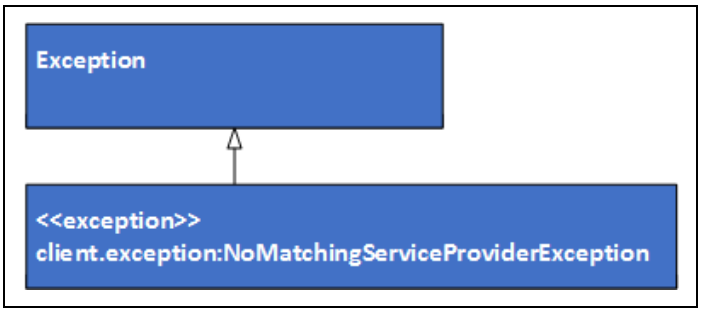


Fig. 14: UML class diagram NoMatchingServiceProviderException exception

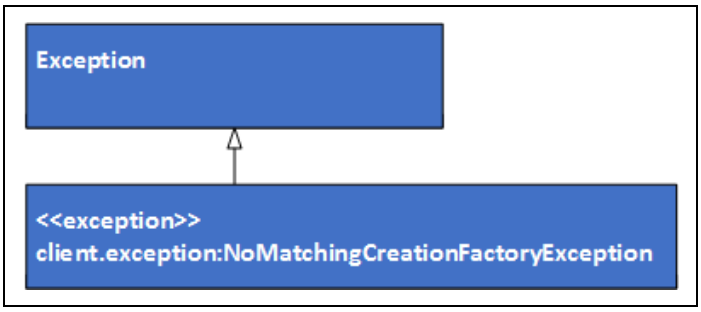


Fig. 15: UML class diagram NoMatchingCreationFactoryException exception

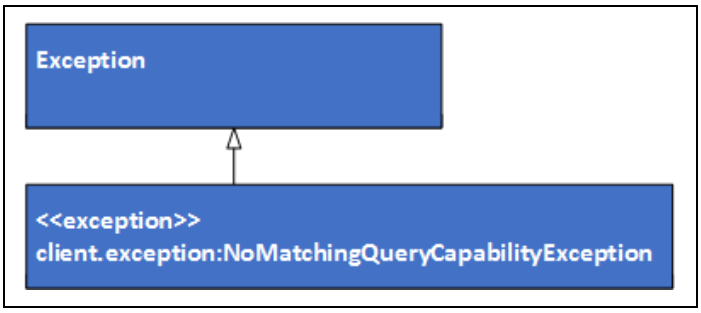


Fig. 16: UML class diagram NoMatchingQueryCapabilityException exception

9.5 Project rtc_oslc_provider

This project includes the persistence for the server. This includes the interface *IPersistence*, the implementing class *NonPersistentStorageSingleton* and the *ResourceHandler* class to modify resources at creation. It also includes the main part of the server consisting of the *ServerApplication* and the *ServletListener*. These are used by the surrounding framework to make up the core of the server implementation. The project also includes the *ServiceProviderSingleton* to be used as the central service provider for the server. In

addition the project finally includes the *TestCaseService* to handle requests send to the server that are used to operate on TestCase resources.

9.5.1 Components

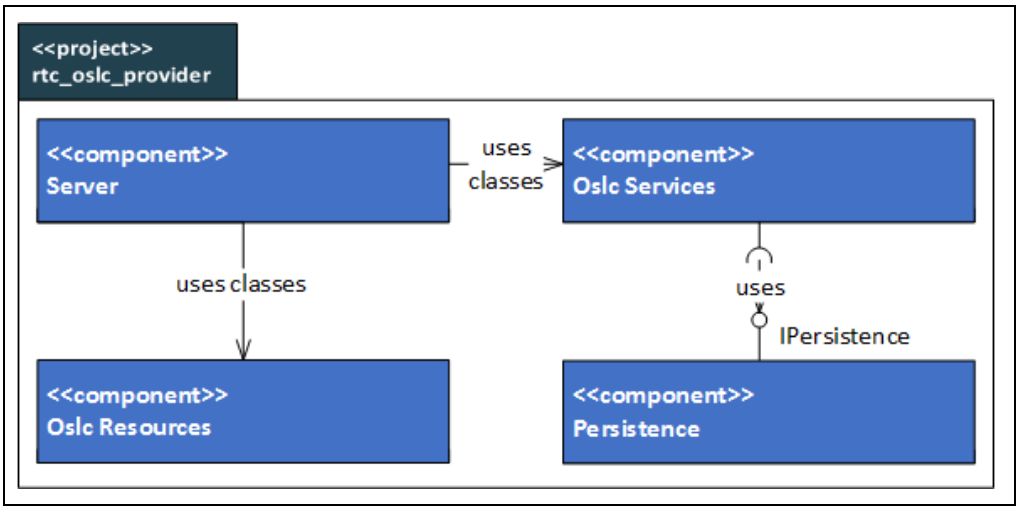


Fig. 17: UML component diagram "rtc_oslc_provider"-project (external view).

9.5.2 OSLC Resources Component

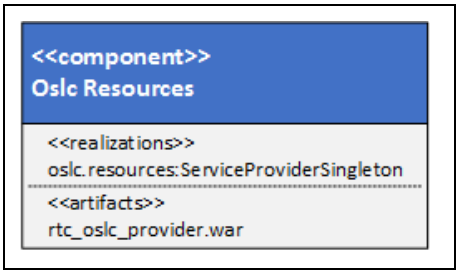


Fig. 18: UML component diagram Oslc Resources component (internal view).

This is included in a separate component because it includes the service provider resource which is used in the server. This resource is not included in the *"rtc_oslc_common"*-project because the service provider resource must not be available for creation and manipulation in the client (see Fig. 18, page 53).

This component is represented by the "*oslc.resources*"-package. It includes the *ServiceProviderSingleton* class.

The *ServiceProviderSingleton* is a singleton. To achieve this it includes a field of the type *ServiceProvider*called*instance* and a *getInstance* method. The type *ServiceProvider* is included in the Lyo SDK libraries. The *getInstance* method returns the value of the *instance*

field or if it is not already bound it sets the value. This includes creating a *ServiceProvider* and binding it to the *instance* field as well as setting OSLC properties for it. In addition it sets all the prefixes for the service provider for the different namespaces.

The class is a singleton because the server must only use one service provider which is handling the OSLC QM resources.

9.5.3 OSLC Services Component

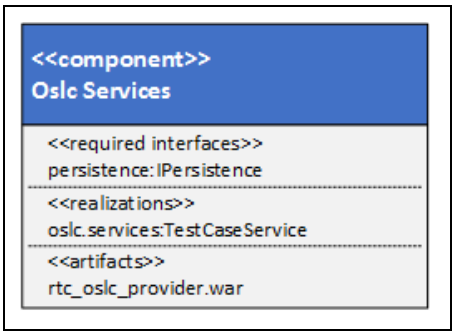


Fig. 19: UML component diagram Oslc Services component (internal view).

This is included in a separate component because it includes the functionality of the resource services for the provider. This component is represented by the *"oslc.services"*-package. It includes the *TestCaseService* class (see Fig. 19, page 54).

The *TestCaseService*class includes methods for handling requests from the client. The methods are *getTestCases*, *getTestCase*, *addTestCase*, *updateTestCase* and *deleteTestCase*.

The *getTestCases* method is used for handling query strings to return matching test cases while the *getTestCase* is used for returning the test case with the matching resource identifier. Both methods include the GET annotation to specify them as Http GET methods.

The *addTestCase* method is used for handling test case creation. It handles client requests to the creation factory URI to create resources. The test case will be added to the persistence. The method includes the POST annotation to specify it as Http POST method.

The *updateTestCase* method is used to update test cases. It handles client requests sent to the location of a test case. The properties of the test case will be updated with the properties of the test case given as parameter. The method includes the PUT annotation to specify it as Http PUT method.

The *deleteTestCase* method is used to delete test cases. It handles client requests send to the location of a test case. The test case will be deleted from the persistence and excluded from the server. The method includes the DELETE annotation to specify it as Http DELETE method.

9.5.4 Persistence Component

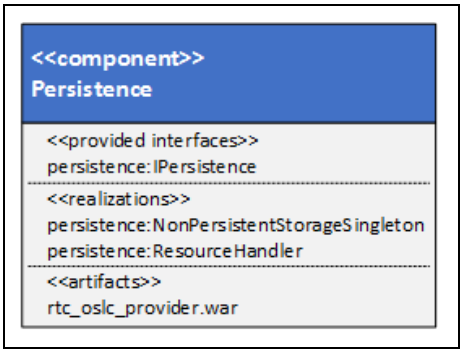


Fig. 20: UML component diagram Persistence component (internal view).

This is included in a separate component because it includes the functionality of the persistence for the server. This component is represented by the "*persistence*"-package. It includes the *ResourceHandler* and *NonPersistentStorageSingleton* class as well as the *IPersistence* interface (see Fig. 20, page 55).

The *IPersistence* is used to specify the interface for the server's persistence. The interface includes methods for adding *addResource*, updating *udpateResource* and deleting resources *deleteResource*. It also includes two methods for resource retrieval. The first is a bulk operation *getResources* while the second can be used to retrieve a single resource *getResource*.

The *NonPersistentStorageSingleton* class is implementing the *IPersistence* interface and provides the functionality for the five specified methods. The *NonPersistentStorageSingleton* is a singleton. To achieve this it includes a field of the type *NonPersistentStorageSingleton* named *instance* and a *getInstance* method. The *getInstance* method returns the value of the *instance* field or if it is not already bound it sets the value. The storage included in the prototype adaptor is non-persistent because it does not create added value for the prototype.

The *ResourceHandler* class includes the *setResourceIdentifier* used in resource creation to set the identifier property of the resource. Therefore the class includes a field for every OSLC QM resource type that holds the identifier value that was last used. This is always updated by the *setResourceIdentifier* to avoid doubling the identifier values in different resources.

9.5.5 Server Component

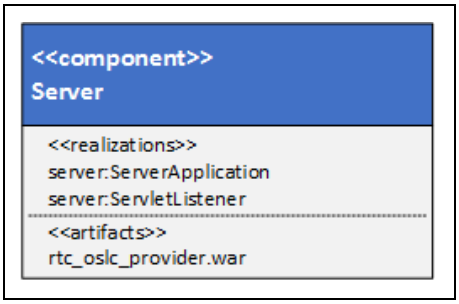


Fig. 21: UML component diagram Server component (internal view).

This is included in its own component because it includes the functionality for the server. This includes the server application itself and a servlet listener. This component is represented by the "*server*"-package. It includes the *ServerApplication* and *ServletListener* class (see Fig. 21, page 56).

The *ServerApplication* class uses *OslcWinkApplication* from the Lyo SDK library as a superclass. It only includes a constructor that passes on the values of the *RESOURCE_CLASSES* and *RESOURCE_SHAPE_PATH_TO_RESOURCE_CLASS* fields. The *RESOURCE_CLASSES* field includes the providers for JENA and JSON4J and includes the *TestCaseService.class* value. The *RESOURCE_SHAPE_PATH_TO_RESOURCE_CLASS* includes the test case path and the *TestCaseService.class* value in a map entry.

The *ServerApplication*is run by the surrounding framework as the main part of the server system and includes the servlet listener for creating and registering a service provider. It also uses the *TestCaseService* class to handle the specific requests created by the client included in the Lyo SDK library.

The *ServletListener* class uses the superclass *ServletContextListener* to initialize *contextInitialized* and destroy *contextDestroyed* the context of the application.

The *ServletListener* class includes an inner class *RegistrationTask* which includes a run task to create a service provider and register it with the OSLC4J Registry.

9.5.6 Class Diagrams

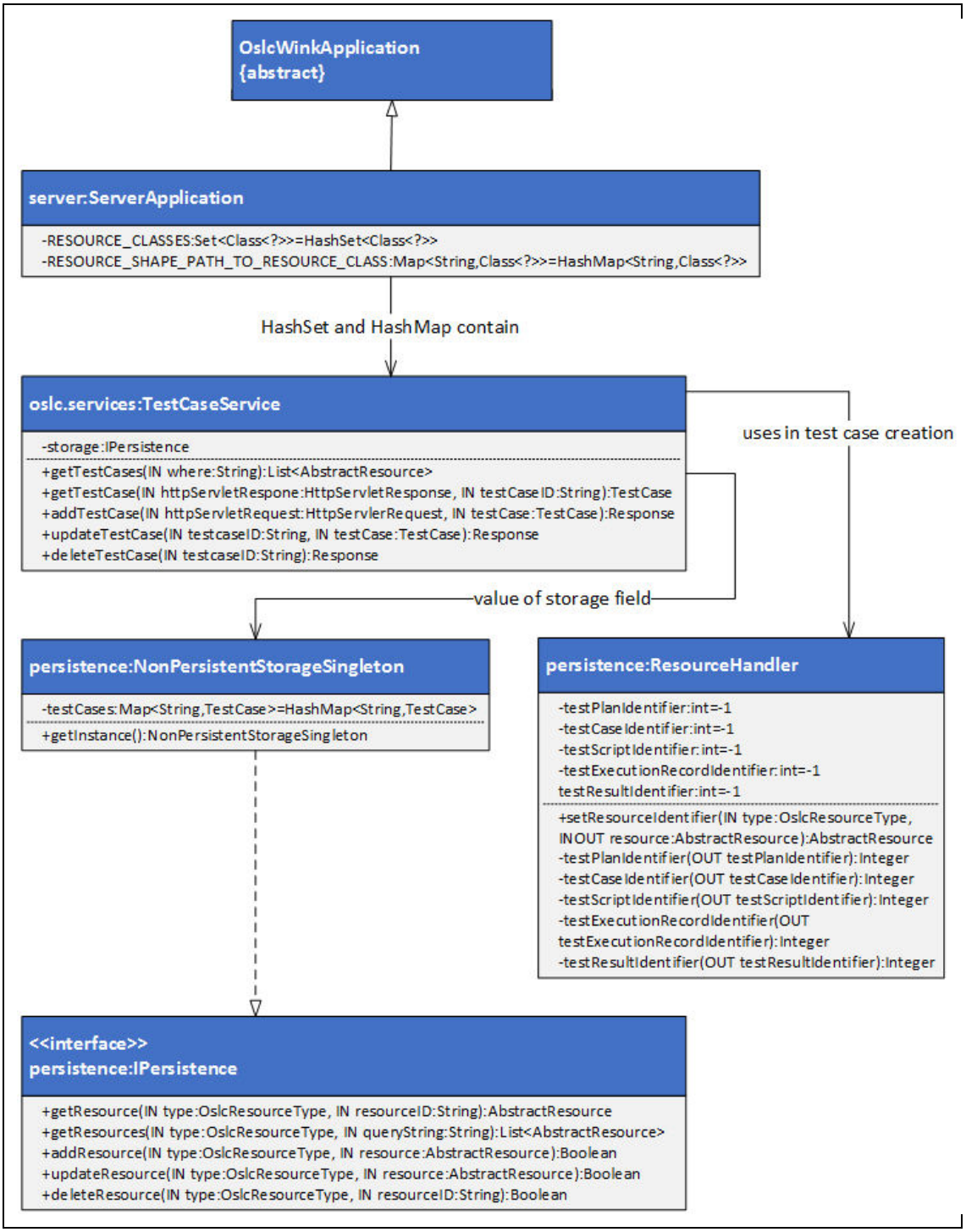


Fig. 22: UML class diagram TestCaseService, NonPersistentStorage, ResourceHandler, ServerApplication classes

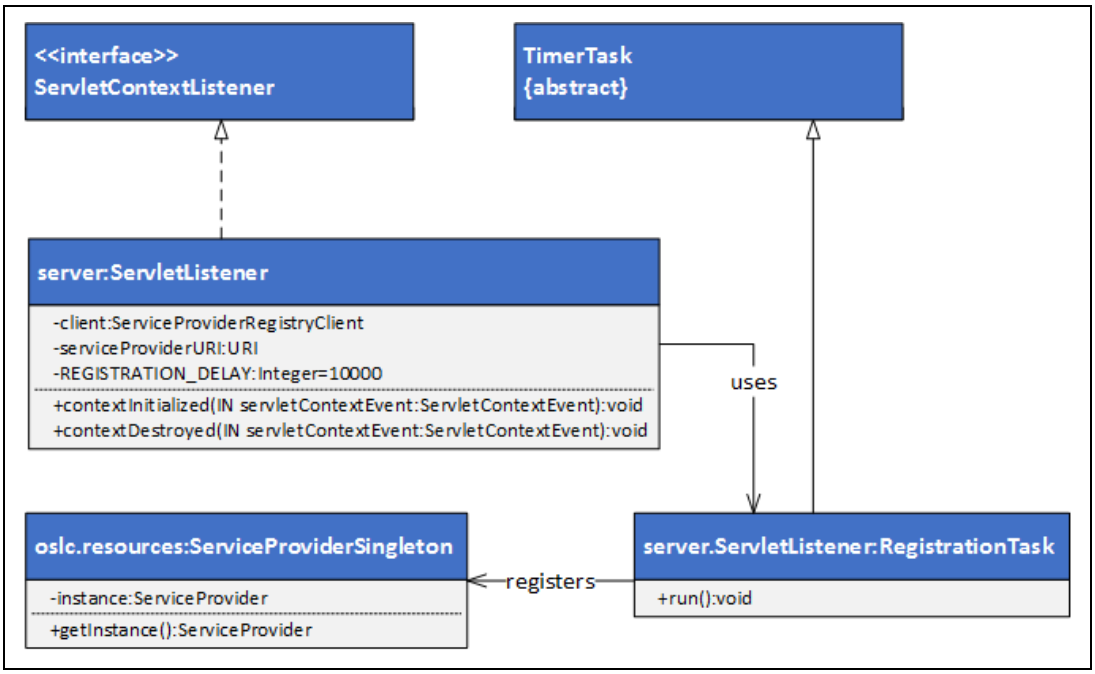


Fig. 23: UML class diagram ServletListener, ServiceproviderSingleton, RegistrationTask classes

9.6 Project rtc_oslc_common

This project includes the OSLC specific *Constants* class that includes the constants used in the other two projects. In addition is also includes the implementation of the OSLC QM resources (*TestCase*, *TestExecutionRecord*, *TestPlan*, *TestResult, TestScript*) and the *Person* resource from the OSLC core. The project finally includes the *OslcResourceType* enumeration which is used for switching between the different behaviors demanded by the different resources.

9.6.1 Components

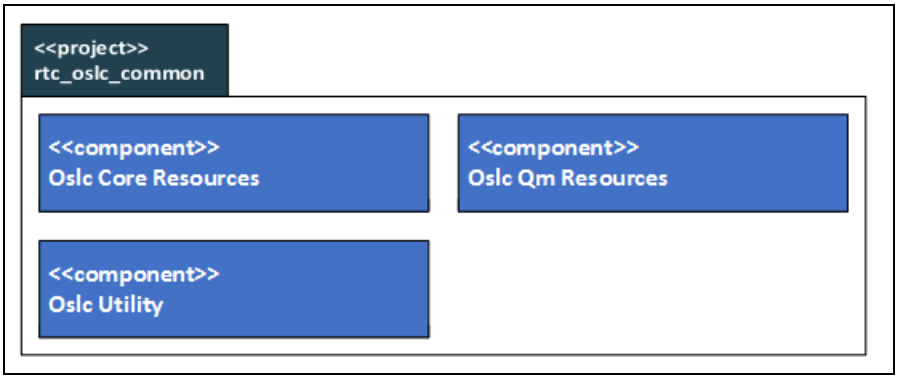


Fig. 24: UML component diagram "rtc_oslc_common"-project (external view).

9.6.2 OSLC Core Resources Component

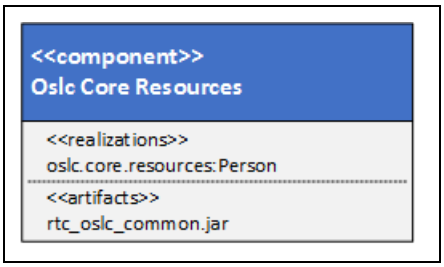


Fig. 25: UML component diagram Oslc Core Resources component (internal view).

This is included in a separate component because the included resource belongs to the OSLC core. This component is represented by the "*oslc.core.resources*"-package. It includes the *Person* class (see Fig. 30, page 62) which implements the OSLC Core resource with the same name (see Fig. 25, page 59).

The resources include the properties and relational properties specified in the corresponding OSLC specification as fields. The fields are named after their respective OSLC properties and are implemented with the type specified by the specification. The properties that have a cardinality of zero-or-many or one-or-many are implemented as lists. These fields include add as well as delete methods for single list entries in addition to the getters and setters that all fields possess.

An example for the field *name* is the *setName* and *getName* methods. An *addName* method is not included because the name is not implemented as a list field.

The getters include OSLC specific annotations from OSLC4J. These include the *title*, *name*, *description*, *propertyDefinition* and *occurs* annotations.

9.6.3 OSLC QM Resources Component

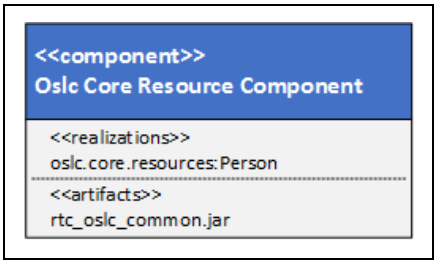


Fig. 26: UML component diagram Oslc Qm Resources component (internal view).

This is included in a separate component because the included resources belong to the OSLC QM domain. This component is represented by the "*oslc.qm.resources*"-package. It includes the *TestCase*, *TestExecutionRecord, TestPlan*, *TestResult,* and *TestScript* class (see Fig. 31 - Fig. 35, page 62 - 64) which implements the OSLC QM resources of the same name (see Fig. 26, page 60).

The implementation of the OSLC QM resources follows the same principles that are described in the “OSLC Core Resource Component” section (see 9.6.3, page 60).

9.6.4 OSLC Utility Component

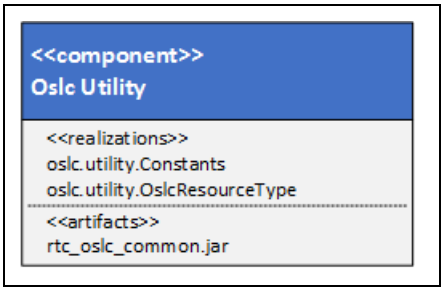


Fig. 27: UML component diagram Oslc Utility component (internal view).

This is included in a separate component because it includes functionality that is separate from the resource implementations included in the other components. This component is represented by the "*oslc.utility*"-package. It includes the abstract *Constants* class and the *OslcResourceType* enumeration (see Fig. 27, page 60).

The constants included are the OSLC QM domain, namespace and namespace prefix constants. In addition it includes the type and path constants for test cases.

The *OslcResourceType* enumeration includes TESTPLAN, TESTCASE, TESTSCRIPT, TESTEXECUTIONRECORD and TESTRESULT as a value. The enumeration is used for different methods in the other projects. It is needed to have one general method declaration and

only switch the different behavior inside the method. This allows avoiding duplicating code as well as hiding the different behavior outside of the classes that implement the functionality.

9.6.5 Class Diagrams

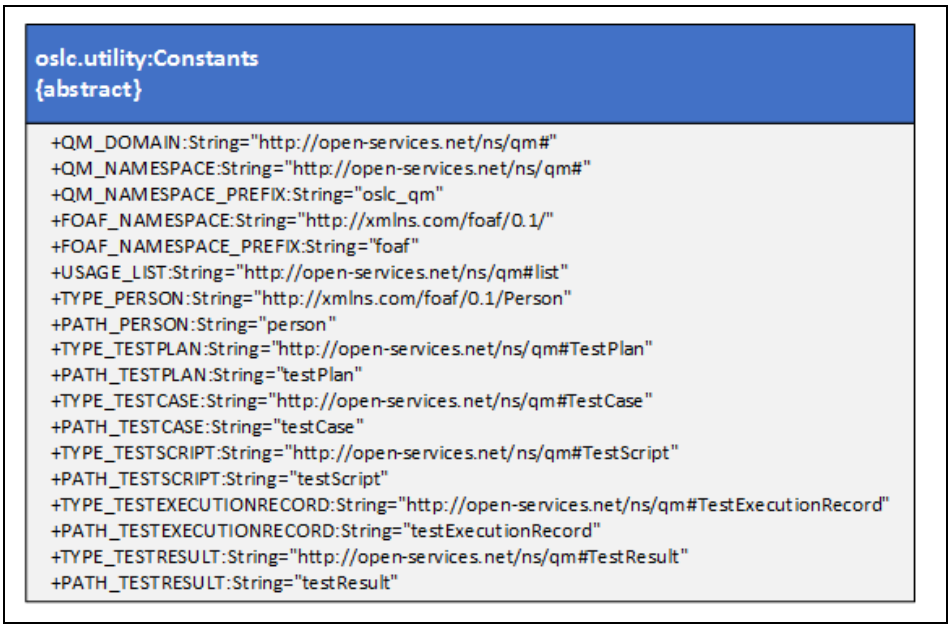


Fig. 28: UML class diagram Constants class

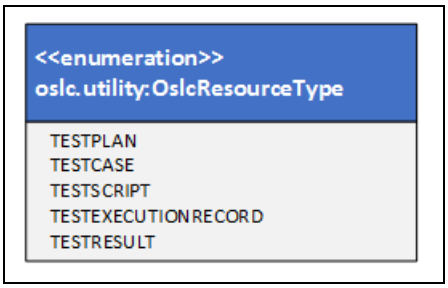


Fig. 29: UML class diagram OslcResourceType enumeration

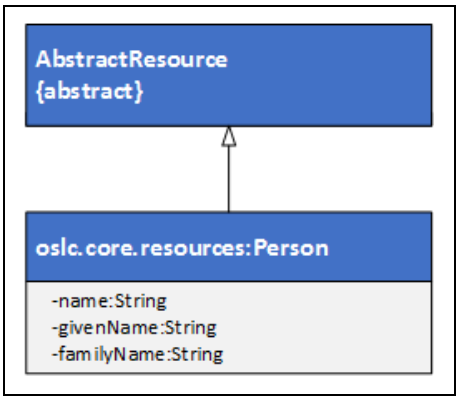


Fig. 30: UML class diagram

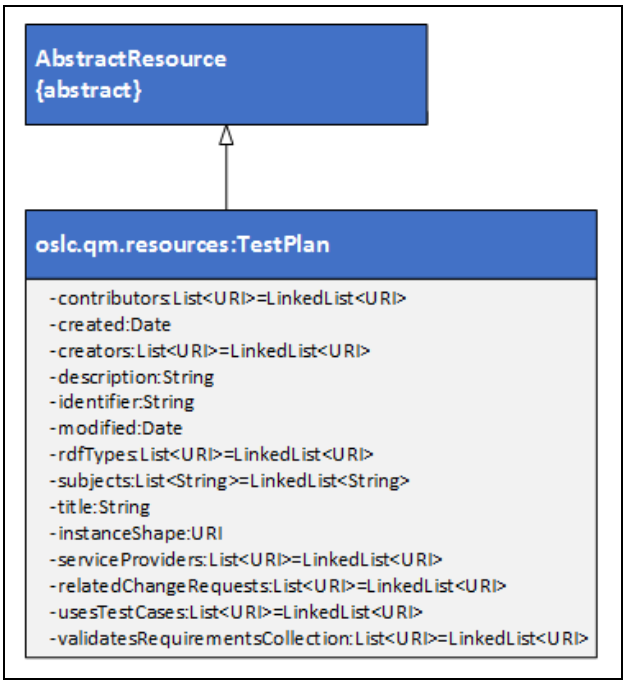


Fig. 31: UML class diagram

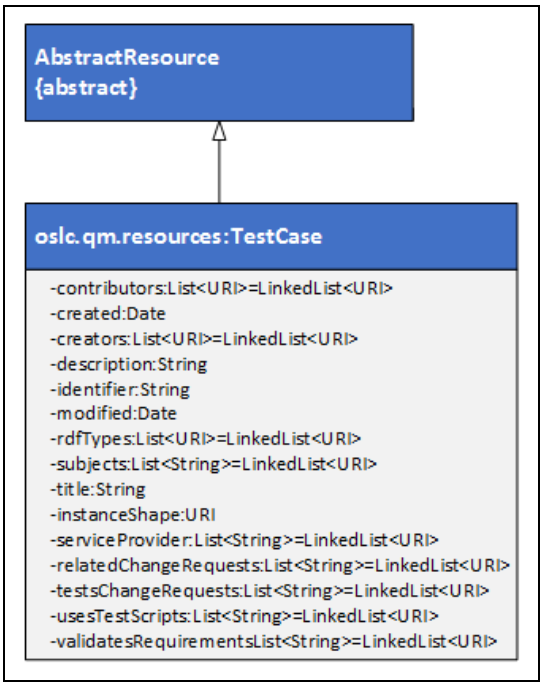


Fig. 32: UML class diagram

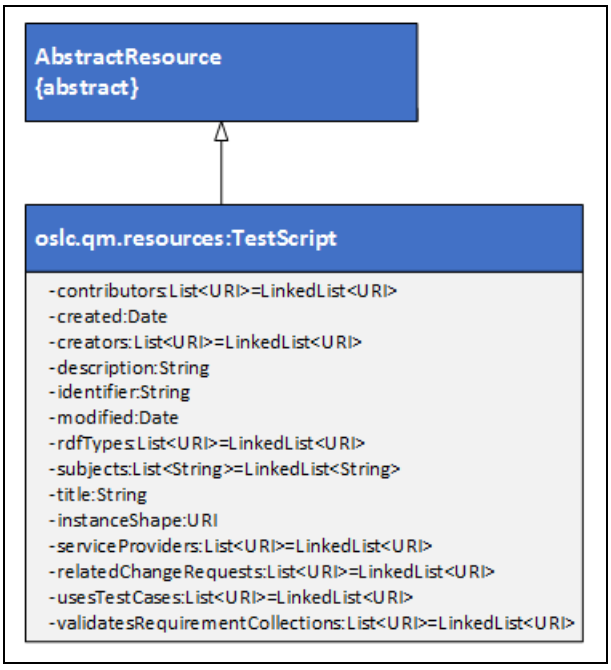


Fig. 33: UML class diagram

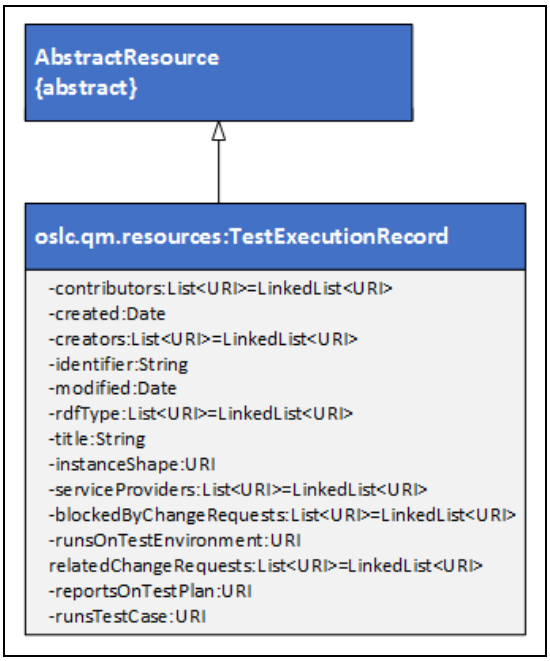


Fig. 34: UML class diagram

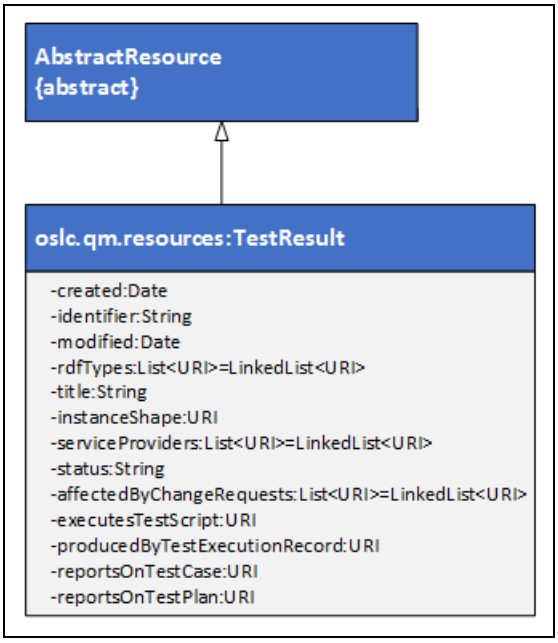


Fig. 35: UML class diagram

9.7 Scenario Execution

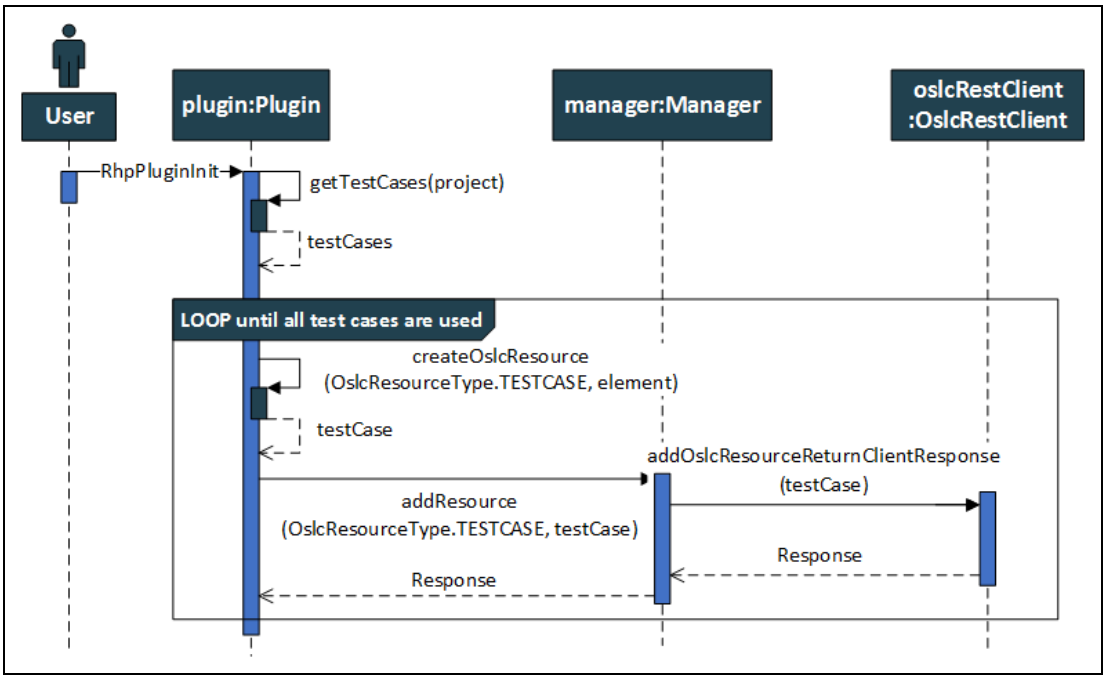


Fig. 36: UML sequence diagram scenario client-side

The scenario requires the collaboration of several methods to provide the needed operations to fulfill the different steps.

This starts in the *"rtc_oslc_adaptor"*-project with the *Plugin* class and the method *RhpPluginInit*. This method calls the *getTestCases* method with the currently opened RTC project as argument. The method then retrieves all *testCases* included in the currently opened project and returns them in the form of a list. Every element of the list is then used in the same steps. These steps include the *createOslcResource* method with the element argument as well as the *OslcResourceType.TESTCASE* value. This is followed by the *addResource* call on the instance of the *Manager* class that is included in the *Plugin* class. In the Manager instance the *OslcRestClient* instance is created to use *addOslcResourceReturnClientResponse* with the *testCase* argument to send it to the provider. This is the last step included in the *rtc_oslc_adaptor* project. The HTTP based communication between the client and server projects is handled by the Lyo SDK.

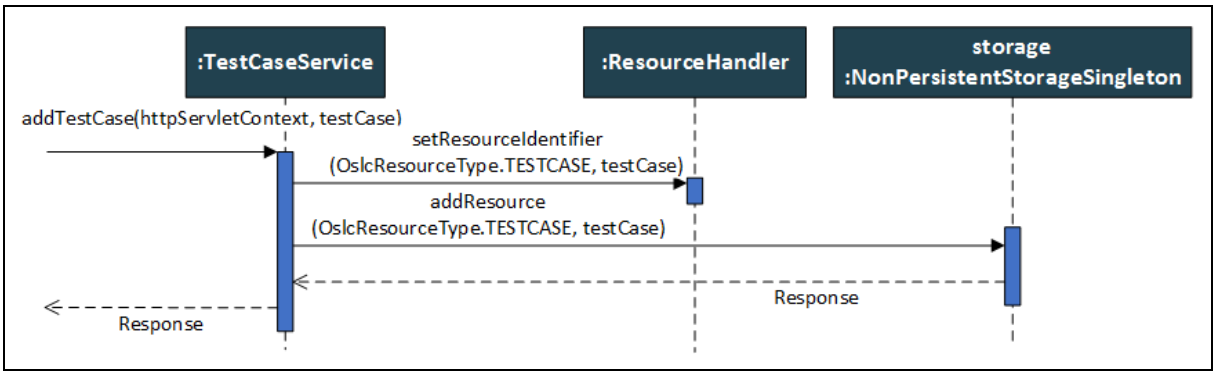


Fig. 37: UML sequence diagram scenario server-side

At this point the *rtc_oslc_provider* project is used. The *addTestCase* method of the *TestCaseService* class is called as an HTTP POST method. The included *testCase* argument is used in the *setResourceIdentifier* method of the *ResourceHelper* class. This method sets the identifier of the *testCase*. This is necessary because the identifier must only be manipulated on the server side therefore they do not include an identifier when they are sent. Finally the *testCase* argument is used in the *addResource* method of the *NonPersistentStorageSingletion* instance that is used in the *TestCaseService* class and returns the *reponse* object to the client side. This concludes the server-sided steps for the scenario.

10 Implementation

Programming Language	
Java	Java in Version 1.6

Development Environment	
Eclipse EGit Plugin	Eclipse Plugin, Pure Java Git integration for Eclipse, used for source code management of OSLC4J
IDE	"Eclipse IDE for Java EE Developers" Juno edition
m2e Plugin	Eclipse Plugin, Maven integration for Eclipse, used for dependency management and building the project

Dependency	Description
rhapsody.jar	Java API for "Rational Rhapsody
Lyo SDK	The Java based Software Development Kit for OSLC implementation The Lyo SDK provides the functionality as OSLC4J libraries. These were included in the project by checking out the source code of the projects with the "Eclipse EGit Plugin". Afterward the "m2e Plugin" is used to generate .jars from the projects to include them as dependencies.

11 Conclusion

My findings are based on my theoretical work with OSLC as well as my experience with my prototype implementation. The OSLC specification includes many different aspects and domains but I will only focus on the OSLC Core and the Quality Management domain as I have worked with them closely.

The conclusion includes the outcome of my work regarding the coverage of my requirements and the scenario. In addition the conclusion also includes the different aspects of OSLC. It is divided into three parts. The first part is about the theoretical aspects of OSLC including the specification and the goal which is achieving interoperability as well as traceability. The second part includes the more general aspects which include the tutorials and the philosophy of OSLC. The third part focuses on the practical aspects of OSLC including the Lyo SDK and the aspect of mapping resources.

11.1 General Appraisal

OSLC as a technology does satisfy the requirements to achieve interoperability and traceability. It is a high quality specification using the latest in available technology and techniques.

The Lyo SDK hinders OSLC on a practical level because it is not on the same high level as the specification. If the Lyo SDK would be improved this would change and improve the practical aspects of using OSLC.

The fact that OSLC is a very new technology reduces the amount of available information on the topic. This currently hinders the efficient and effective use of OSLC but will also improve automatically over the course of time as these sources of information become available.

11.2 Coverage of Requirements

The table below includes the analysis of the requirements in the aspect of the implementation.

Requirement	Explanation	Satisfied
Traceability	Satisfied by mapping the RTC constructs to OSLC resources and making them available on the provider.	✓
Interoperability	Satisfied because the provider and adaptor of the implementation can interact with other adaptors and providers. These have to be implemented using the Lyo SDK or implement the HTTP accessible methods listed in the “Interoperability” section (see 11.5, page 71).	✓
Specification	Satisfied because the Lyo SDK is used that is closely checking if the created resources are correctly fulfilling the specification.	✓
Simplicity	Satisfied because the implementation does provide the minimal set of functionality that is needed for the scenario.	✓
Modularity and Extensibility	Satisfied by the architecture of the different components.	✓

11.3 Coverage of Scenario

The following table includes the list of the steps of the scenario and if they were satisfied by the prototype implementation.

Scenario Steps	Satisfied
The adaptor reads in the test cases of the project.	✓
The adaptor creates OSLC QM test cases based on the retrieved test cases.	✓
The adaptor posts the OSLC QM test cases to the provider.	✓
The provider makes these OSLC QM test cases accessible by OSLC consumers.	✓
The provider makes these OSLC QM test cases accessible by a browser.	✓

The following table includes the list of the post conditions of the scenario and if they were satisfied by the prototype implementation.

Scenario Post Conditions	Satisfied
Every project test case is represented by an OSLC QM test case.	✓
Every OSLC QM test case is present on the provider.	✓
Every OSLC QM test case on the provider is accessible by OSLC consumers.	✓
Every OSLC QM test case on the provider is accessible via browser.	✓

11.4 Specification

The OSLC specification is listing many different aspects of its inner workings which include resources, properties as well as services to name the most important ones. The specification is using an intuitive way of describing these aspects. Two examples of this are included below.

Prefixed Name	Occurs	Read-only	Value-type	Representation	Range	Description
OSLC Core: Common Properties						
dcterms:contributor	zero-or-many	unspecified	Either Resource or Local Resource	Either Reference or Inline	any	Contributor or contributors to resource (reference: Dublin Core). It is likely that the target resource will be an foaf:Person but that is not necessarily the case.
dcterms:created	zero-or-one	True	DateTime	n/a	n/a	Timestamp of resource creation (reference: Dublin Core)

Fig. 38: Example for the specification of common properties of OSLC resources

This is an example for an OSLC resource. In this case an OSLC QM TestCase and its properties (see Fig. 38, page 70). The table entry first specifies the name of the included property followed by important information and finishes with a short description of the property. This is a very clean and simple way of describing resources and properties.

Prefixed Name	Occurs	Read-only	Value-type	Representation	Range	Description
Relationship properties: This grouping of properties is used to identify relationships between resources managed by OSLC Service Providers						
oslc_qm:relatedChangeRequest	zero-or-many	False	Resource	Reference	any	A related change request. It is likely that the target resource will be an oslc_cm:ChangeRequest but that is not necessarily the case.

Fig. 39: Example for the specification of relationship properties of OSLC resources

This is an example for the description of a relationship property (see Fig. 39, page 70). It basically is described in the same way as the OSLC resources but is included in a different section to highlight that it is a relational property and not a common property. Dividing this into different sections is very useful because the relational properties are very important.

A problem in the specification is that it relies heavily on the word SHOULD for describing many aspects. This leads to the problem that a high amount of the specification could be ignored as SHOULD marks the content as not mandatory. Identifying a minimal working set of mandatory aspects that form OSLC is a time consuming and non-trivial task.

Another problem is that it misses a top down perspective of explaining OSLC. This would include a very abstract view of OSLC and what it is, a rough outline of its inner working as well as what are the basics of what can be achieved by applying OSLC.

11.4.1 Impact

The above mentioned influenced my work in regard to needing a lot of time to grasp the basic concepts of OSLC because the top down perspective was lacking for my purposes. This was also the case because I could not make out the minimal working set that forms OSLC which made acquiring the needed knowledge time consuming. But the intuitive, simple and clean way to present the information of the specific resources and the properties did help on a more detailed level.

11.4.2 Appraisal

I cannot explicitly say if the problems I had while working with the OSLC specification are a hindrance to people who have more experience in this field. From my personal position the OSLC specification is still good enough but needs to be improved in regard to these aspects.

11.5 Interoperability

OSLC shows promise in creating interoperability capabilities in different tools. OSLC allows to a certain degree for syntactic interoperability in tools by providing a common data format for the different constructs included in a tool. This format is represented in the form of using the OSLC resources and the properties that are specific for them.

The syntactic level of interoperability is only achieved to a certain degree because the OSLC resources can be exchanged between different tools but they cannot provide the same level of information that is included in the original constructs they are based on.

But the exchanged level of information is sufficient because it allows the basic information of constructs to be exchanged across different tools. For more specific information the tool construct that the OSLC resource is based on must be looked at.

This is acceptable because it is impossible to find data formats for every construct in every tool that correctly maps all information. In addition it must be used and processed by every possible tool. This would create complex data that must be maintained.

The above mentioned is the complete opposite of what interoperability should be which is easy and fast to use and understand as well as efficient and effective.

One exception has to be made from applying interoperability by using OSLC with the Lyo SDK. If the other tools do not provide a well formed interface that can be accessed using

the general HTTP (POST, PUT and GET) methods interoperability cannot be achieved. Therefore the other tool has to correctly implement the specification to provide the interoperability capabilities.

11.5.1 Impact

As stated above I did not find any problems while using OSLC to create interoperability capabilities for "Rational Rhapsody Test Conductor" therefore this did not have any impact on my work.

11.5.2 Appraisal

The lack of finding any problems while using OSLC to create tool interoperability leads me to the conclusion that OSLC is very well suited for that purpose. OSLC cannot be used to achieve semantic interoperability between different tools but this is not a goal of OSLC and can therefore not be seen as a lacking feature. Nonetheless OSLC would be the wrong technology if the goal is to achieve syntactic and semantic interoperability.

11.6 Traceability

OSLC shows promise in creating traceability capabilities in different tools. The different constructs of tools can be transformed into OSLC resources which can then be traced throughout the lifecycle as well as across domains and across different tools. This is the case because the different OSLC resources can be held in the providers which are accessible regardless of the specific tool constructs they are based on.

The fact that the constructs of a tool are not published directly but only indirectly as OSLC resources creates the need to keep them synchronized. This means that only the simplified OSLC representation of the tool constructs is worked with and not the original.

11.6.1 Impact

The need to keep constructs of the tool and OSLC resources synchronized was no influential factor for my prototype implementation as it is not a fully functional business solution. The implementation of the prototype does not use persistent data storage for the mapped resources. A business solution must use the provided interface to connect to a database. The synchronization of the connected database is automatically provided by the adaptor.
The need to synchronize the data is not an OSLC specific requirement but a general requirement for different technologies. Therefore this is not a disadvantage for OSLC.

11.6.2 Appraisal

OSLC can be applied to create traceability capabilities in an effective and efficient manner which makes using it a good choice.

11.7 Tutorials and Examples

Tutorials and examples are a very important aspect of every technology as it enables people to use the technology efficiently and effectively. Therefore I included them together in one section. I differentiate between tutorials and examples at the point that a tutorial has to explain in great detail how the person doing the tutorial can achieve the objective while the example is just about showing how something should be.

The examples that are included in OSLC are good regarding quality and quantity. There are several examples for resources for the different domains as well as the core. Additionally the high quality of the specification I mentioned in the “Specification” section (see 11.4, page 70) does not leave much need for examples for most things as the description in the specification itself is sufficient.

The tutorials for OSLC and how to implement OSLC are lacking. The tutorials are not explicit enough for explanation and are often stuck on a very abstract level which is not sufficient to qualify as an actual tutorial. Most of these tutorials are good as abstract examples of what can be done with OSLC and what the advantages of applying OSLC are. But they do not give a detailed step by step explanation how to actually apply OSLC. In addition some of the code based implementation tutorials are just example projects. These include an explanation how to setup the projects and run them but do not explain the actual functionality of the project. Another problem of some tutorials and examples regarding the implementation of OSLC is that they are not efficient enough in the way they provide explanation.

11.7.1 Impact

The above mentioned problems lead to the problem that I needed a lot of time to actually understand the implementation of OSLC. This can cause an additional problem because understanding the steps that need to be undertaken to implement OSLC can become a resource consuming task in both time and money. The tutorial I actually used to implement OSLC was the script of a workshop I participated in that was held in the context of the MBAT project. The problem with this script is that it is not available outside of the MBAT project and I could only participate in the workshop and get the script because the bachelor thesis was created in the context of the MBAT project.

11.7.2 Appraisal

The above mentioned problems lead me to the conclusion that some of the OSLC tutorials should be renamed as examples because they do not provide necessary explanations. In addition the tutorials that include example implementations should be enriched with explanations how the implementation was created and what the core aspects are. Additionally the script from the workshop should be included together with the provided example projects in the OSLC wiki to help with the implementation of OSLC.

11.8 Resources and Mapping

The mapping of constructs to OSLC resources involves three steps. The first is choosing which OSLC resource the construct is equivalent to. The second involves the actual mapping process of creating the OSLC resource and setting its properties. The third and last step should then include checking if additional properties need to be created for the OSLC resource to reflect the important information of the basic construct that need to be checked directly in the OSLC resource. The third step was not tested in my work as the prototype I implemented is only serving of a general prove of concept.

The difficulty of the first step depends on the number of resources included in the OSLC domain that is used. In the OSLC QM domain there are five resources which makes matching the tools constructs to these resources not too complicated. The only problem that might occur is that the wording in OSLC sometimes overlaps with the vocabulary used in computer science but differs in the definition of the meaning.

The second step is intuitive because most properties of OSLC resources are reflecting information that is included in most constructs anyway. These include title, description, time stamps for creation and modification as well as links to the creators and contributors of the resource.

The number of resources and the coverage of different domain aspects for the OSLC QM domain are easy to work with aside from the above mentioned problems. The low number of resources allows for an efficient mapping process while still providing the ability to map all important information without creating additional resources. This is also support by the low number of properties that are used in the OSLC resources which leads to a good coverage of useful information while not providing the need to map unnecessary details. This makes OSLC resources very efficient and effective to use.

11.8.1 Impact

The first step for the mapping process is rather complex depending on the tool that is used and how many constructs it supports. But this is not the case because OSLC is used as a technology but is based on the complexity of the tool used.
The second step was straight forward because it only involved using given information of the resources in the properties of the OSLC resources. This task can be done fast and easily.

The fact that the wording of OSLC is duplicating vocabulary while not having the same meaning can make the process more complex. This depended on the fact that I always needed some time to change the perspective to differing meaning of the vocabulary.

11.8.2 Appraisal

In my opinion the mapping process of OSLC is intuitive and can be done in a short amount of time. The resources are well fit for the mapping process but the explanations of the content a resource should contain should be a little more detailed and maybe enriched

with some small examples. This would make the process of understanding which tool constructs are equivalent to which OSLC resources easier.

11.9 Lyo Software Development Kit (SDK)

The Lyo SDK can be used to implement OSLC solutions. It does have advantages as well as problems. Advantages include that OSLC resources can be implemented as Plain Old Java Objects (POJOs) which makes handling the resources easier while also providing the option to set the different representational types like rdf/xml or json.

One of the problems of the Lyo SDK is that it is not documented and explained well enough which hinders fast and efficient work with it. This is the case because basic tasks that are supported by the Lyo SDK need a lot of time to understand and use even though they should not be as complex and hard to understand. In addition there are not enough sources that include explanation of the Lyo SDK and its functionality.

In addition it does not go far enough in helping with the implementation. This means that the OSLC resources have to be implemented with the help of the Lyo SDK. An example for this is implementing all the OSLC QM resources but they are based on the specification which leaves no room for interpretation. Therefore these resources should already be implemented by the Lyo SDK and be available for direct use from the library or as superclass if there is the need to add custom properties.

This is discussed in the Lyo development mailing group now (June 2013). The implementation of OSLC resources might be included in a later version of the Lyo SDK. At the moment this is not the case therefore this is still a problem.

Another problem of the Lyo SDK is that constructs that are known from other technologies and implementation feel unintuitive to use. I picked the OSLC Client as an example for this.

```
OslcRestClient oslcRestClient = new OslcRestClient(providers, queryBase, MEDIA_TYPE, timeout);
TestCase[] testCase = oslcRestClient.getOslcResources(TestCase[].class);
```

Fig. 40: Code snippet of the OslcRestClient included in the Lyo SDK

In the first line of the example is the constructor for the OSLC4J Rest Client. Included is the variable *queryBase* which specifies the URI of the query service. In the second line the OSLC4J Rest Client is used to retrieve resources in this case a test cases. The resources are retrieved from the URI that was specified in the constructor.

This is not intuitive because every time another resource needs to be retrieved the *queryBase* variable must be set to a new destination before using the actual method for retrieval. Additionally the use of the "class" method in java is very uncommon.

A more intuitive approach would be to specify the URI for the query service as a parameter in the *getOslcResources* method so that every method call can include a new *queryBase* value.

11.9.1 Impact

The Lyo SDK heavily influences the effort needed for the implementation. Working with the Lyo SDK the first time was very time consuming. A lot of this time was spent using examples and tutorials from the workshop as well as participating in the "DevCafe" workshop[9] to understand how to use and apply the SDK. The workshop helped greatly in understanding the Lyo SDK. Unfortunately the workshop is not publicly available.

In addition the Lyo SDK did not generate added value for my implementation. This might partly depend on the fact that it is a prototype but it is also the case because of the above mentioned problems of the Lyo SDK.

11.9.2 Appraisal

In my opinion the problems I have mentioned above need to be solved because it influences the quality of the Lyo SDK in a negative way. Developers must be able to efficiently and effectively use the SDK or otherwise it will not be useful for supporting OSLC which could lead to the lack of implementations and usage of the technology.

11.10 Implementation Problems

11.10.1 Maven and Lyo SDK

Maven is a very powerful tool which makes it hard to use at first. After focusing on the basic functionality of the "m2e Plugin" it worked very well. The plugin saves a lot of time by automatically handling the dependency management for the project. In addition it allowed using the Lyo SDK because manually integrating the OSLC4J libraries in the project becomes obsolete. The problem I had using the "m2e Plugin" initially was based on the fact that it checks the local repository saved on the hard disk before it checks the general maven apache repository. The Lyo SDK .jars are not integrated in the general maven apache repository and are therefore not available. This was obfuscated because I ran an example implementation earlier that included the Lyo SDK .jars in the local repository which were missing for my own projects. To solve this problem I used the "Eclipse EGit Plugin" to check out the source code of the Lyo SDK. The projects containing the Lyo SDK can then be transformed into .jars with the "m2e Plugin" and added to the project as dependency.

[9] Paschke 2013

11.11 Outlook

The prototype implementation can be upgraded to a fully functional business solution by applying changes in some places.

11.11.1 Using all relevant Constructs of Rational Rhapsody Test Conductor

To use all RTC constructs that are specified as relevant for mapping in the "Analysis" section (see 8, page 41) the prototype must be changed in some places. The first step is including the read in and resource creation in the *Plugin* class. These can be closely modeled according to the already existing methods and source code sections already included for the test cases. The next step includes adding the source code for the different resource types specified by the *OslcResourceType* enumeration in the *Consumer* and *Manager* classes that implement the *IResourceConsumer* and *IResourceManager* interfaces.

In addition the steps of the "Resource Services" section (see 11.11.2, page 77) must be satisfied. Finally the changes of the "Persistent Database" section (see 11.11.3, page 77) should be included as well. No other changes have to be made to include and use all relevant RTC constructs.

11.11.2 Resource Services

The missing OSLC QM resource services (*TestPlan*, *TestScript*, *TestExecutionRecord*, *TestResult*) have to be implemented for the provider. They can be closely modeled according to the already existing *TestCaseService* class. This step will provide the ability to use these resources in the provider. The newly created resource services need to be added to the *RESOURCE_CLASSES* field of the *ServerApplication* class. The path and type constants have to be added to the constants class of the "*rtc_oslc_common”*-project.

No other changes are necessary.

11.11.3 Persistent Database

The prototype does not include persistent data storage. Persistent data storage has to be applied for a fully functional solution. To support the persistent data storage, the *IPersistence* interface is provided and must be used. The persistent data storage has to replace the *NonPersistentStorageSingleton* class.

No other changes are necessary to include a persistent storage for the resources of the provider.

11.11.4 Synchronization

The synchronization of the RTC constructs with OSLC QM resources is already given. This is the case because the *Plugin* will always update the OSLC QM resources whenever the base construct from RTC is changed. The only part missing is the functionality for synchronizing deleted RTC constructs. This has to be implemented.

No other changes are necessary to make sure the RTC constructs are synchronized with the OSLC QM resource counterparts.

11.11.5 Plugin Integration for Rational Rhapsody Test Conductor

To create a plugin that can be called from the RTC the *“rtc_oslc_adaptor”* and “*rtc_oslc_common”*-projects have to be compiled into a single jar. This is needed to include all the dependencies to make them available for the RTC. The jar has to be included into the rhapsody.ini to load the plugin when the RTC is started.

No other changes are necessary to create an integrated plugin for the RTC.

12 Glossary

In the table below are the terms used in this paper that are not commonly known terms which need further explanation. This excludes terms like HTTP or XML because these are commonly known and are not special terms used in this paper.

Term	Definition
API	Application Programming Interface. Library to interact with the system that provides the API.
Artifact	Every product created during the software development cycle.
Consumer	The OSLC term for a client that retrieves OSLC resources from an OSLC provider
EGit	Git plugin for eclipse. Used for checking out the Lyo SDK source code from the repository
IDE	Integrated Development Environment. Application that provides several capabilities to create software
Interoperability	Ability of different tools or systems to interact and communicate with each other
Linkeddata	Data that does not only include information but is also linked to other sources of data to increase the value of the information
Lyo SDK	The software development kit for OSLC. Supports the creation of OSLC based applications
M2E	Maven plugin for eclipse. Used for build automation and dependency management
ModelBus	Framework for the creation of integrated tool environments
OSLC	Open Services for Lifecycle Collaboration. Specification to allow integration of tools

Term	Definition
OSLC AM	Open Services for Lifecycle Integration Architecture Management. Architecture domain workgroup of OSLC, includes domain specific resources
OSLC QM	Open Services for Lifecycle Integration Quality Management. Quality domain workgroup of OSLC, includes domain specific resources.
OSLC RM	Open Services for Lifecycle Integration Requirement Management. Requirement domain workgroup of OSLC, includes domain specific resources.
OSLC4J	Toolkit for creating OSLC based applications. Supports creation of consumers and providers as well as enables easy communication for these constructs
Property	Information in an OSLC resource.
Provider	The OSLC term for a server that provides OSLC resources
Rational Rhapsody Test Conductor	Integrated tool for scenario based testing in the UML based software development tool Rational Rhapsody
Resource	Artifact representation in OSLC
SDK	Software Development Kit. Software that helps in creating applications for specific systems or technologies
Service	Operation on an OSLC resource
Test Case	OSLC QM resource
Test Execution Record	OSLC QM resource
Test Plan	OSLC QM resource
Test Result	OSLC QM resource
Test Script	OSLC QM resource
Traceability	Ability to follow and track artifacts throughout their lifecycle. Used for gaining additional information about the artifacts and enabling additional techniques like change management and impact analysis

13 Bibliography

Arwe 2013
Arwe, John: Open Services for Lifecycle Collaboration, Core Specification Version 2.0. URL: http://open-services.net/bin/view/Main/OslcCoreSpecification (8 January 2013)

Brockmeyer 2013
Brockmeyer, Udo; Lettrari, Marc; Rothaupt, Ruben; Wachtendorf, Christian: Workshop Implementation of Rational Rhapsody Test Conductor (RTC) Plug-in (7/8 March 2013). Oldenburg: BTC Business Technology Consulting AG, 2013

Dai 2012
Dai, Zhen Ru: ISTQB Certified Tester, Foundation Level (CTFL). Lecture, summer semester 2012 at Hamburg University of Applied Sciences (HAW). Hamburg: 2012.

Fiedler 2013-1
Fiedler, Michael: Lyo. Ottawa (Canada): Eclipse Foundation, 2013. URL: http://wiki.eclipse.org/Lyo (4 April 2013)

Fiedler 2013-2
Fiedler, Michael: Building and Running Lyo OSLC4J applications in Eclipse. Ottawa (Canada): Eclipse Foundation, 2013. URL: http://wiki.eclipse.org/Lyo/BuildingOSLC4J (4 April 2013)

Mailing list Lyo
Lyo project developer discussions, lyo-dev@eclipse.org

Mailing list OSLC
OSLC Core, oslc-core@open-services.net

McMahan 2011
McMahan, Paul: Open Services for Lifecycle Collaboration, Quality Management Specification Version 2.0, Terminology. 2011. URL: http://open-services.net/bin/view/Main/QmSpecificationV2#Terminology (10 January 2013)

OSLC 2013-1
Open Services for Lifecycle Collaboration: About. 2013. URL: http://open-services.net/about/. Note: This page is no longer available on the current (June 2013) OSLC web site, but it is put into the archives by "The Wayback Machine" http://archive.org/web/web.php (see Appendix C: page 87).

OSLC 2013-2
Open Services for Lifecycle Collaboration. 2013. URL: http://open-services.net/. (10 January 2013)

Paschke 2013
Paschke, Stefan: Online Workshop MBAT RTPv1 Dev Café, Implementation of OSLC Consumer and Provider using the Lyo SDK. 5/25 April 2013

Ritter
Ritter, Tom (responsible editor): Overview, Features. München: Fraunhofer-Gesellschaft zur Förderung der angewandten Forschung e.V. URL: http://www.modelbus.org/modelbus/index.php/overview/features (14 March 2013)

14 Appendix

Appendix A: Installation Guide

Rational Rhapsody Test Conductor

1. Install Rational Rhapsody Version 8.0 from https://www14.software.ibm.com/webapp/download/preconfig.jsp?id=2012-09-21+05%3A17%3A50.571065R&S_TACT=&S_CMP=. Select all programming languages in the installation. Install the Test Conductor add-on when asked to install plug-ins and add-ons in the installation process.

Eclipse

1. Install Eclipse IDE for Java EE Developers Juno Edition.
2. Install EGit plug-in. Add http://download.eclipse.org/egit/updates as Eclipse update page to install the plug-in.
3. Install M2E plug-in. Add http://download.eclipse.org/technology/m2e/releases as Eclipse update page to install the plug-in.

Setup Project and Dependencies

1. Import > Existing Projects > Archive File *rtc_oslc_implementation* and import "*rtc_oslc_common"*, "*rtc_oslc_adaptor"* and "*rtc_oslc_provider"*.
2. Follow the below mentioned steps on http://wiki.eclipse.org/Lyo/BuildingOSLC4J[10]:
   - "Clone the Lyo Core and RIO(optional, for Change and Quality Management samples) git repositories
   - "Import OSLC4J Eclipse Projects from the git repository"
   - "Build all projects"

[10] Fiedler 2013-2

3. Right click *pom.xml* from *rtc_oslc_common*, *rtc_oslc_adaptor* and *rtc_oslc_provider* and *Run As>Maven clean* for each project.
4. Right click *pom.xml* from "*rtc_oslc_common*", "*rtc_oslc_adaptor*" and "*rtc_oslc_provider*" and *Run As>Maven install* for each project. If "Build Success" shows in the console the project was successfully build, if not check alternative step 4b.
5. Open *Run Configurations* and select the *Plugin* class' *RunConfiguration*. In the *Arguments* tab add "-Djava.library.path=C:\tools\IBM\Rational\Rhapsody\8.0\Share\JavaAPI" as *VM argument*. This has to point to the Rational Rhapsody Installations JavaAPI.
6. Run the *rtc_oslc_provider Maven build*. When the console states the error "INFO: RegistrationTask:ServiceProvider registered." run the Plugin class.

Alternative Step

4b. If the *rtc_oslc_adaptor* build is not successful check the console for missing symbols warning. Right click *pom.xml* and *Run As>Maven clean*. Right click the project and select *Build Path>Configure Build Path*. In the *libraries* tab remove the rhapsody.jar. Afterwards right click them pom.xml and *Run As>Maven install*. After the installation run return to *Build Path Configuration* and click *Add Jars* in the *libraries* tab. Select the *rhapsody.jar* from the *rhapsody* folder of the *lib* folder in the *rtc_oslc_adaptor* project.

Launch Configuration and Running the Project

Run the *Plugin* class and the *rtc_oslc_provider* run maven configuration. If they do not exist follow steps 1-5.

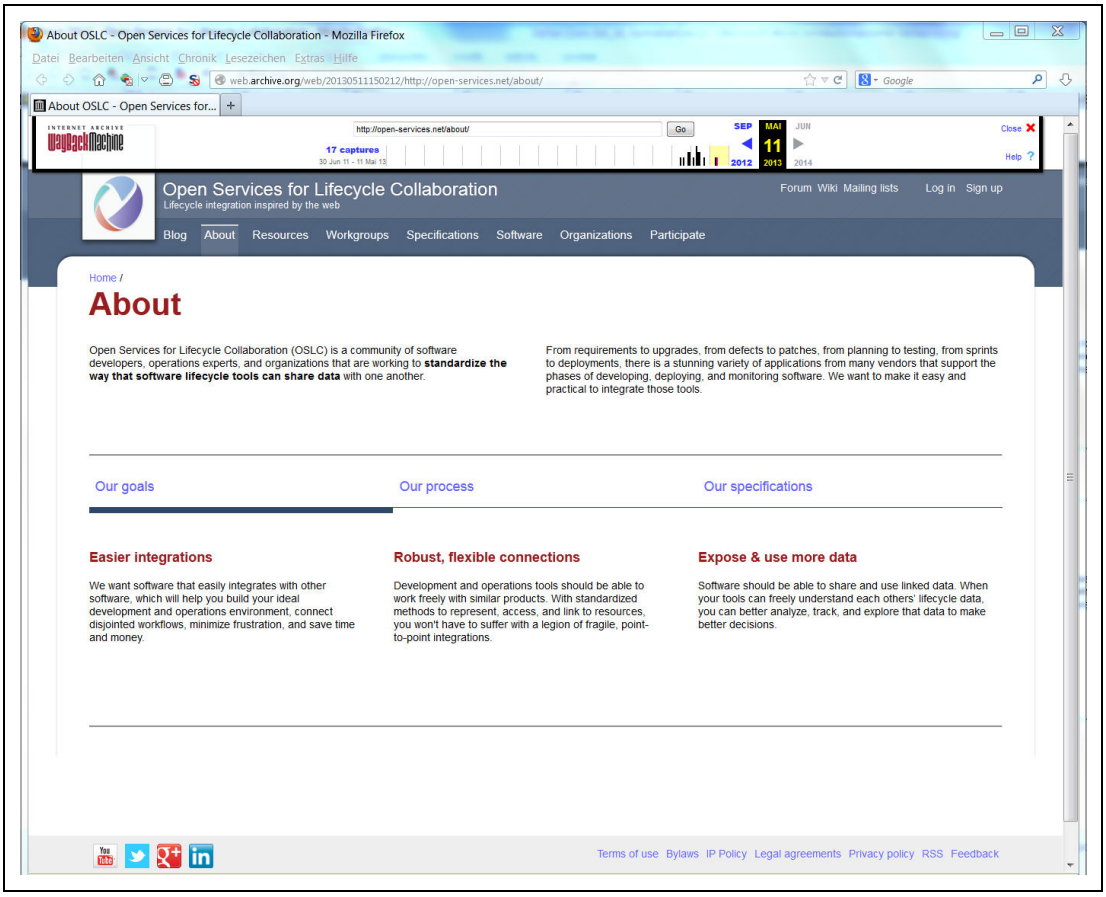
1. Open *Run Configurations* and select the *Plugin* class' *RunConfiguration*. In the *Arguments* tab add "-Djava.library.path=C:\tools\IBM\Rational\Rhapsody\8.0\Share\JavaAPI" as *VM argument*. This has to point to the Rational Rhapsody Installations JavaAPI.
2. Open *Run Configurations* and create a new configuration under the *Maven* section. Set the B*ase Directory* to *${workspace_loc:/rtc_oslc_adaptor}* in the main tab. The *Goal* has to be *jetty:run-exploded*.
3. In the *Source* tab add "*rtc_oslc_common"*, "*rtc_oslc_provider"* and *OSLC4JRegistry.*
4. In the *Environment* tab add a variable *OSLC4JREG* with the *Value* *${workspace_loc:/OSLC4JRegistry}*
5. In the *common* tab under saved as select *Shared File* and set it to *\rtc_oslc_provider\test\launches*

Appendix B: Table of Figures

Note: To create the figures 1-7 I used Microsoft Power Point 2010, and Microsoft Visio Professional 2013 for the figures 8-37.

Appendix C: Archived Website

The following figure shows the former "About OSLC" page of the OSLC Community web site (archived on 11 May, 2013).



Source: http: //web.archive.org/web/20130511150212/http://open-services.net/about/

Versicherung über Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, den _____