DEGREE PROJECT IN ELECTRICAL ENGINEERING, SECOND CYCLE
STOCKHOLM, SWEDEN 2017

Push-based low-latency solution for Tracked Resource Set protocol

An extension of Open Services for Lifecycle Collaboration specification

XUFEI NING

KTH ROYAL INSTITUTE OF TECHNOLOGY
INFORMATION AND COMMUNICATION TECHNOLOGY

Push-based low-latency solution for Tracked Resource Set protocol

An extension of Open Services for Lifecycle Collaboration specification

Xufei Ning

2017-08-28

Master's Thesis

Examiner
Gerald Q. Maguire Jr.

Academic adviser
Jad El-Khoury

KTH Royal Institute of Technology
School of Information and Communication Technology (ICT)
Department of Communication Systems
SE-100 44 Stockholm, Sweden

Abstract

Currently, the development of embedded system requires a variety of software and tools. Moreover, most of this software and tools are standalone applications, thus they are unconnected and their data can be inconsistent and duplicated. This increase both heterogeneity and the complexity of the development environment.

To address this situation, tool integration solutions based on Linked Data are used, as they provide scalable and sustainable integration across different engineering tools. Different systems can access and share data by following the Linked-Data-based Open Service for Lifecycle Collaboration (OSLC) specification. OSLC uses the Tracked Resource Set (TRS) protocol to enable a server to expose a resource set and to enable a client to discover a resource in the resource set.

Currently, the TRS protocol uses a client pull for the client to update its data and to synchronize with the server. However, this method is inefficient and time consuming. Moreover, high-frequency pulling may introduce an extra burden on the network and server, while low-frequency pulling increases the system's latency (as seen by the client).

A push-based low-latency solution for the TRS protocol was implemented using Message Queue Telemetry Transport (MQTT) technology. The TRS server uses MQTT to push the update patch (called a ChangeEvent) to the TRS client, then the client updates its content according to this ChangeEvent. As a result, the TRS client synchronizes with the TRS server in real-time.

Furthermore, a TRS adaptor was developed for Atlassian's JIRA, a widely-used project and issue management tool. This JIRA-TRS adaptor provides a TRS provider with the ability to share data via JIRA with other software or tools which utilize the TRS protocol.

In addition, a simulator was developed to simulate the operations in JIRA for a period of time (specifically the create, modify, and delete actions regarding issues) and acts as a validator to check if the data in TRS client matches the data in JIRA.

An evaluation of the push-based TRS system shows an average synchronization delay of around 30 milliseconds. This is a huge change compared with original TRS system that synchronized every 60 seconds.

Keywords

Linked Data, Open Services for Lifecycle Collaboration, Tracked Resource Set, JIRA, Message Queue Telemetry Transport, Push Technology

Sammanfattning

Nuvarande inbyggda system kräver en mängd olika program och verktyg för att stödja dess utveckling. Dessutom är de flesta av dessa programvara och verktyg fristående applikationer. De är oanslutna och deras data kan vara inkonsistent och duplicerad. Detta medför ökad heterogenitet och ökar komplexiteten i utvecklingsmiljön.

För att hantera denna situation används verktygsintegrationslösningar baserade på Länkad Data, eftersom de ger en skalbar och hållbar integrationslösning för olika tekniska verktyg. Olika system kan komma åt och dela data genom att följa den Länkad Data-baserade tjänsten Open Service for Lifecycle Collaboration (OSLC). OSLC använder TRS-protokollet (Tracked Resource Set) så att en server kan exponera en resursuppsättning och för att möjliggöra för en klient att upptäcka en resurs i resursuppsättningen.

TRS-protokollet använder för tillfället pull-metoden så att klienten kan uppdatera sin data och synkronisera med servern. Denna metod är emellertid ineffektiv och tidskrävande. Vidare kan en högfrekvensdriven pull-metod införa en extra börda på nätverket och servern, medan lågfrekvensdriven ökar systemets latens (som ses av klienten).

I det här examensprojektet implementerar vi en pushbaserad låg latenslösning för TRS-protokollet. Den teknik som används är Message Queue Telemetry Transport (MQTT). TRS-servern använder MQTT för att pusha uppdateringspatchen (som kallas ChangeEvent) till TRS-klienten. Därefter uppdaterar klienten dess innehåll enligt denna ChangeEvent. Vilket resulterar i att TRS-klienten synkroniseras med TRS-servern i realtid.

Dessutom utvecklas en TRS-adapter för Atlassians JIRA som är ett välanvänt projekt och problemhanteringsverktyg. JIRA-TRS-adaptern tillhandahåller en TRS-leverantör med möjlighet att dela data via JIRA med annan programvara eller verktyg som använder TRS-protokollet.

Dessutom utvecklade vi en simulator för att simulera verksamheten i JIRA under en tidsperiod (specifikt skapa, ändra och ta bort åtgärder rörande problem) och en validator för att kontrollera om data i TRS-klienten matchar data i JIRA.

En utvärdering av det pushbaserade TRS-systemet visar en genomsnittlig synkroniseringsfördröjning på cirka 30 millisekunder. Detta är en stor förändring jämfört med det ursprungliga TRS-systemet som synkroniseras var 60:e sekund.

Nyckelord

Länkade Data, Open Services for Lifecycle Collaboration, Tracked Resource Set, JIRA, Message Queue Telemetry Transport, Push-teknologi

Acknowledgments

I would like to thank my supervisor, Jad El-Khoury, for the technique support throughout my thesis project. And I would like to express my gratitude to my KTH examiner, Gerald Q. Maguire Jr., for his academic guidance. Additionally, I would thank my colleagues in Scania, Andrii Berezovskyi, Daniel Echegaray, Shuang Zheng, Yash Khatri for their help and advice.

Special thanks to my family and girlfriend, who always support me for all these years.

Stockholm, August 2017
Xufei Ning

Table of contents

List of Figures

List of Tables

List of acronyms and abbreviations

API	Application Program Interface
ASSUME	Affordable Safe And Secure Mobility Evolution
GUI	Graphical User Interface
HTTP	Hyper Text Transfer Protocol
JAX-RS	Java API for RESTful Web Services
JQL	JIRA Query Language
JSON	Java Script Object Notation
JSON-LD	Java Script Object Notation for Linked Data
LDP	Linked Data Platform
LQE	Lifecycle Query Engine
MQTT	Message Queue Telemetry Transport
OSLC	Open Services for Lifecycle Collaboration
QoS	Quality of Service
RDF	Resource Description Framework
REST	Representational State Transfer
SDK	Software Development Kit
TCP/IP	Transmission Control Protocol/Internet Protocol
TRS	Tracked Resource Set
UI	User Interface
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
W3C	World Wide Web Consortium
XML	Extensible Markup Language

1 Introduction

This Master's thesis project took place at Scania Tekniskt Centrum in Södertälje, Sweden. It is a part of the ongoing ASSUME [1] research project, which aims to provide affordable, safe, and reliable transport solutions. This chapter introduces the general background needed by the reader of this thesis project, presents the challenges and the problem, and then proposes a practical solution.

1.1 Background

Currently, embedded system development requires a variety of software and tools. Moreover, most of this software and tools are standalone applications, thus they are unconnected and their data can be inconsistent and duplicated. This brings increased heterogeneity and increases the complexity of the development environment.

To address this situation, tool integration solutions based on Linked Data [2] are used, as they provide scalable and sustainable integration across different engineering tools. Different systems can access and share data by following the Linked-Data-based Open Service for Lifecycle Collaboration (OSLC) specification [3]. OSLC use the Tracked Resource Set (TRS) protocol [4] to enable a server to expose a resource set and to enable a client to discover a resource in the resource set.

The TRS protocol is HTTP-based and follows RESTful principles. An example of the TRS architecture is shown in Figure 1-1. JIRA is a project management tool, which can create records for issues with properties such as name, id, creator, time, Uniform Resource Identifier (URI), etc. JIRA's main competitor is Bugzilla [5], OSLC has implemented a TRS provider for Bugzilla, which can be found in [6].

Figure 1-1 shows three issues saved in JIRA together with their different properties: name, id, creator, created time, description, and JIRA URI. An OSLC adaptor is connected to JIRA and configured to share only three properties: name, id, description. In addition, the "OSLC URI" property is added to refer to each of the issues saved by the OSLC adaptor.

A TRS server holds a resource set. The resource set lists members in an OSLC adaptor as a catalog, where these members are identified by OSLC URIs. A TRS client maintains a local copy of the resource set, then accesses each OSLC URI to create an initial copy of each issue and saves the results in a local database (in this case in a Triple Store). Both TRS servers and TRS client do not store any data from the resources, but rather request this data as they need it.

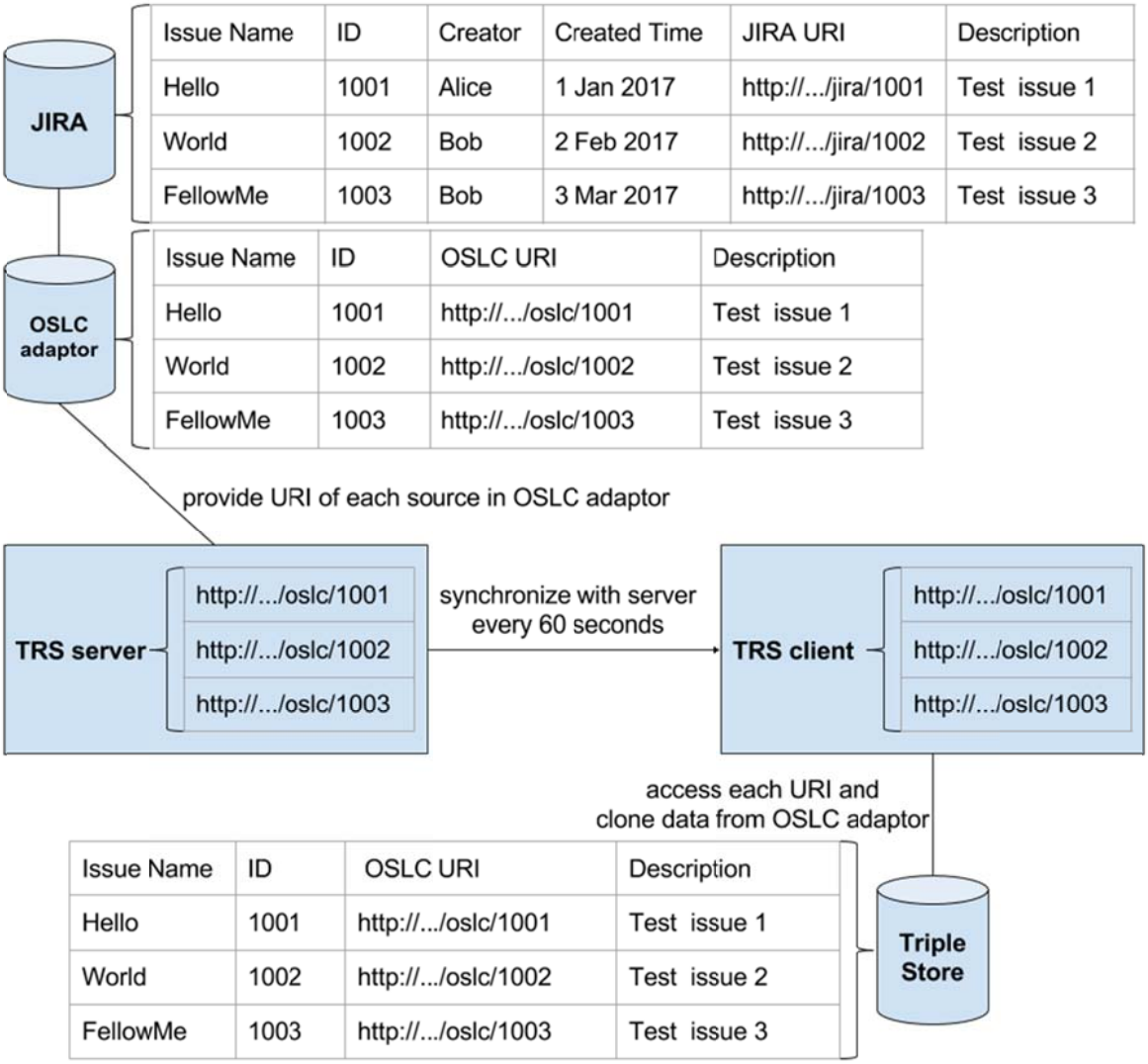

Figure 1-1: TRS Architecture

1.2 Problem Statement

Currently, the TRS protocol client performs periodic pulls to update its data and to synchronize with the server. A TRS client sends a pull request to the TRS server every 60 seconds - whether there is any update or not. This leads to an obvious shortcoming as a client may take more than 60 seconds to process all updates [4]. Additionally, this inevitable leads to “empty pulls” where there is no actual data to exchange.

The current pull-based TRS architecture is inefficient and time consuming, thus making it hard to achieve low-latency without a high polling overhead. Moreover, frequent polling could waste system resources and add an unnecessary burden on the network and server. In contrast, low-frequency polling may increase the latency of synchronization. In some scenarios, such as in a safety critical system, a few-seconds delay may result in a system failure.

A push-based solution that could provide low-latency for TRS is highly desirable for many applications [7].

According to Scania's case study, a push-based TRS architecture was proposed while developing Linked-Data-based information integration [8]. During the Organization for the Advancement of Structured Information Standards (OASIS) Telecon meeting on 2016.06.09 [9], a new approach was proposed where the TRS system could send push-based notifications using the Message Queue Telemetry Transport (MQTT) protocol.

However, this push-based notification is imperfect in real situations. When the information in a notification is limited, the TRS client still needs to send HTTP GET requests to the TRS server to synchronize with server. A better solution is needed [10].

Moreover, for Scania's case study, the TRS system is expected to use JIRA as the data source. However, this case study did not suggest how to develop a TRS provider for JIRA, hence this development needed to be investigated.

1.3 Purpose

This project investigates the architecture of a TRS system, builds a TRS provider for JIRA, proposes and compares possible solutions for push-based TRS architecture, and implements and evaluates a push-based TRS system using practical technology.

Using this push-based TRS architecture, a TRS client will update its content (almost) immediately when changes are made at the TRS server, while reducing the load on the server and reducing network traffic. This is a major optimization compared with the default TRS architecture, which calls the pull method every 60 seconds, whether there is any updated data or not.

Given a JIRA-TRS provider, JIRA's TRS adaptor can share data with other tools or software which also follow the TRS standard.

1.4 Goals

This thesis project entails the design, development, prototyping, and evaluation of a push-based TRS architecture and a TRS provider for JIRA. This goal can be divided into the following four sub-goals:

1) A case study of the TRS protocol and development of a TRS provider for JIRA.
2) A case study of real-time message protocols, leading to a proposal and comparison of practical solutions relevant for this project.
3) Developing and prototyping of a push-based TRS architecture.

4) Development of a JIRA simulator to simulate the operation of the proposed solution for a period of time. This simulation will serve as a validator and enables an evaluation of the performance of the proposed push-based TRS system.

1.5 Research Methodology

To realize the goals two research methods are used in this project:

- The system design part follows design science for the development of the TRS provider and TRS consumer. This project also needed to analysis the TRS protocol and TRS architecture, to design a TRS server algorithm for JIRA, and provides service for both a human operated web browser and computer request.
- The research follows a qualitative research method, with a focus on finding a suitable solution for a push-based TRS architecture. There are several possible technologies that could be used to realize a push-based system, this project selects and compares only the most commonly used ones.

1.6 Delimitations

In this project, there were several choices made to bound the scope of the project:

1) For this prototype of a push-based TRS architecture only one TRS server and client pair are used. We explicitly chose **not** to implement a large-scale use case.
2) We assume all data are transmitted correctly, i.e., without network errors or data loss. As a result, we have not performed any experiments or otherwise investigated how network quality of service (QoS) and network status affect the entire system.

1.7 Structure of the thesis

Chapter 2 presents the theoretical background and related works to help readers to better understand this thesis. Chapter 3 introduces the design and implementation of a TRS provider for JIRA, together with details of the development of a TRS server. Chapter 4 compares several possible technologies that might be used to solve the problem stated in Section 1.2. The most practical of was selected for prototyping. Chapter 5 introduces the design and implementation of a push-based TRS system, including modifications of both the TRS server and client along with serialization and deserialization of updates. Chapter 6 evaluates the performance of the prototype JIRA-TRS provider and push-based TRS architecture, using both a simulator and validator. Chapter 7 presents the conclusion and suggests future work.

2 Background

This chapter provides basic background information about Linked Data, OSLC, TRS, and JIRA. Additionally, this chapter describes related work concerning JIRA adaptors and TRS clients.

2.1 Linked Data

Linked Data is a technology to connect relevant or related data together. The purpose of using linked data is to provide a way for tools and software to share and explore data with other tools [2].

2.1.1 Resource Description Framework

Linked data uses the Resource Description Framework (RDF) to describe data relationships. RDF is a World Wide Web Consortium (W3C) specification. The data relationship described by RDF is called "Triple", as it shown in Figure 2-1 [11].

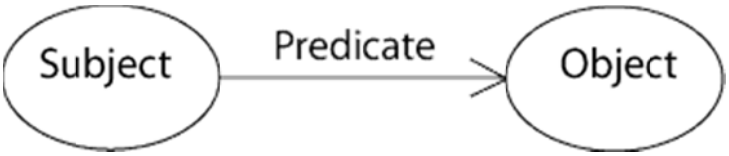

Figure 2-1: RDF Structure

For example, "Alice is a friend of Bob" can also be described in "Triple" format as:

Subject: Alice

Predicate: friend

Object: Bob

A relationship "Issue No.07 created by Admin" converts to the following "Triple":

Subject: Issue No.07

Predicate: creator

Object: Admin

2.1.2 Uniform Resource Identifier

A URI is used to identify a resource. In this project, all of the members in a resource set are describe by unique URIs. For example, "https://example.com/demo_data/HelloWorld.doc_V1.02" identifies a specific version of the document HelloWorld.doc.

2.1.3 Resource set

A resource set contains a collection of resources, and each resource is identified by a URI. For example, a resource set may have members such as the following:

> *Member 1: https://example.com/demo_data/HelloWorld.doc_V1.02*
>
> *Member 2: https://example.com/demo_data/guide.mkv*
>
> *Member 3: https://example.com/requirements/customer.xml_V1.0*

The resource set only contains the URI of each resource and does *not* contain the content of each resource.

2.1.4 Triple Store

A Triple Store is a graph database that stores data in a subject-predicate-object (Triple) format. The Triple Store used in this project is Apache Fuseki [12]. Additionally, this database provides a graphical user interface (GUI) for the user to query for data. The RDF query language is called SPARQL [13].

2.2 Open Services for Lifecycle Collaboration

OSLC is based on linked data. OSLC is an open community that defines practical specifications for integrating software. OSLC aims to provide a free, high-efficiency solution for tool integration [3].

2.2.1 OSLC Adaptor

An OSLC adaptor is used to share data based upon the OSLC specification. An OSLC adaptor can select which data to share when assessed via an OSLC URI.

2.3 Tracked Resource Set

While using the OSLC specification to share data, the data provider may change existing resource relationships or properties. In this case, TRS allows the data provider (i.e., a TRS server) to expose a set of resources and to allow a data consumer (in this case a TRS client) to track all changes (such as data creation,

modification, and deletion) that occur at the data provider (via the OSLC adaptor) [4]. A simple TRS architecture was shown in Figure 1-1. TRS is expressed via the Base and ChangeLog as described below.

2.3.1 Base

Base [4] is a Linked Data Platform (LDP) Container, which lists members in the resource set. Each member is referenced by an URI. A TRS provider may contain an enormous number of members; hence it would be a waste of system resources to regenerate the Base after each change happens. Instead, the Base is designed to be periodically regenerated. A CutoffEvent points to the last update of the Base. All later changes are listed in the ChangeLog.

A Base may be split into multiple pages to provide faster server response. Each page is required to have a reference to indicate the next Base page. If a Base page is the last page, then its next page reference is to "rdf:nil". An example of a TRS Base is shown in Figure 2-2 [3].

```
# Resource: http://cm1.example.com/baseResources/
@prefix trs: <http://open-services.net/ns/core/trs#> .
@prefix ldp: <http://www.w3.org/ns/ldp#> .

<http://cm1.example.com/baseResources/>
  a ldp:DirectContainer;
  ldp:membershipResource <http://cm1.example.com/baseResources/>;
  ldp:hasMemberRelation ldp:member;
  trs:cutoffEvent <urn:urn-3:cm1.example.com:2010-10-27T17:39:31.000Z:101> ;
  ldp:member <http://cm1.example.com/bugs/1> ;
  ldp:member <http://cm1.example.com/bugs/2> ;
  ldp:member <http://cm1.example.com/bugs/3> ;
  ...
  ldp:member <http://cm1.example.com/bugs/199> ;
  ldp:member <http://cm1.example.com/bugs/200> .
```

Figure 2-2: TRS Base

2.3.2 ChangeLog

The ChangeLog [4] contains all the changes as seen by the OSLC adaptor. Each change is called a "ChangeEvent". A ChangeEvent may be one of three types: Creation, Modification, or Deletion. Each ChangeEvent must have an URI. All the ChangeEvents saved in ChangeLog are sorted by time and then tagged with sequence numbers (forming a so-called "change order"). This change order does not need to be consecutive, but a newer change must have a greater order than the previous one.

A ChangeLog may be split into multiple pages. In this case the newer ChangeEvents are in the first few pages. Each ChangeLog page must have a "trs:previous" reference to an earlier ChangeLog page. If a ChangeLog page is the

last page, it also needs a “trs:previous” reference to “rdf:nil”. An example of a TRS ChangeLog is shown in Figure 2-3 [3].

```
# Resource: http://cm1.example.com/trackedResourceSet
@prefix trs: <http://open-services.net/ns/core/trs#> .

<http://cm1.example.com/trackedResourceSet>
  a trs:TrackedResourceSet ;
  trs:base <http://cm1.example.com/baseResources/> ;
  trs:changeLog [
    a trs:ChangeLog ;
    trs:change <urn:urn-3:cm1.example.com:2010-10-27T17:39:33.000Z:103> ;
    trs:change <urn:urn-3:cm1.example.com:2010-10-27T17:39:32.000Z:102> ;
    trs:change <urn:urn-3:cm1.example.com:2010-10-27T17:39:31.000Z:101> ;
    trs:previous <http://cm1.example.com/changeLog/1> .
] .

<urn:urn-3:cm1.example.com:2010-10-27T17:39:33.000Z:103>
...
```

Figure 2-3: TRS ChangeLog

2.4 JIRA

In this project, JIRA is used as a resource provider, as it can create records of issues with properties such as id, project, creator, created time, description, and URI. All these issues with the properties are saved in JSON format [14]. JIRA is a software development tool, mainly used by Agile teams [15].

2.4.1 JIRA REST API

JIRA provides an REST API to access resources via URLs. A user may use the HTTP verbs GET, POST, PUT, and DELETE to interact with JIRA. JIRA will respond to a HTTP request with a response in JSON format [16]. The current RESTful API version is “2”. Alternatively, where requesting this latest version one can use “latest” instead.

2.4.2 JIRA Query Language

The JIRA Query Language (JQL) is used to search for specific issues in JIRA [17]. The default query result is limited to nor more than 1000 results per page. Table 2-1 lists several JQL examples combined with REST.

Table 2-1 JIRA REST API

Search all issues	http://localhost:2990/jira/rest/api/latest/search?
Search total number	http://localhost:2990/jira/rest/api/latest/search?maxResults=0
Set start point	http://localhost:2990/jira/rest/api/latest/search?startAt=0&maxResults=50
Search issue with ID	http://localhost:2990/jira/rest/api/latest/issue=<issue id>

2.4.3 WebHook

WebHook is a callback over HTTP. When using WebHook, a web application sends a HTTP POST when an event happens [18]. JIRA's WebHook is used to send notifications to add-ons or web application when changes made in JIRA [19]. This project used JIRA WebHooks to notify a TRS provider about issue creation, modification, and deletion.

2.5 Related work

There are some previous works relevant to this project. These are primarily contributions are made by KTH Royal Institute of Technology and Scania Group.

2.5.1 JIRA Adaptor

Scania developed an OSLC adaptor for JIRA, which implements the OSLC Change Management specification [20]. This JIRA Adaptor converts JIRA's JSON-format issues into RDF triple format and tags these RDF-triple-format issues with OSLC URIs. Thus, a user may access OSLC URIs to browse JIRA issues in RDF triple format.

2.5.2 TRS Client

A TRS client can synchronize data with a TRS server and save OSLC data in a Triple Store. A well-known and fully developed TRS client is IBM Lifecycle Query Engine (LQE) [21].

Additionally, KTH Royal Institute of Technology has developed a TRS client for research and academic use, which was used in this project. Other researchers or academics who are interested in TRS client development should contact with Jad El-Khoury [22] or Andrii Berezovskyi [23].

2.5.3 Jena Model Helper

To create RDF graphs [24], the Jena Model Helper (a part of project Eclipse Lyo [25]) was used. Eclipse Lyo aims to adopt OSLC specifications and build OSLC-compliant tools. This project uses Jena Model Helper to serialize OSLC resources.

2.6 Summary

An example showing the relationships of the concepts presented earlier is shown in Figure 2-4. In this figure:

- JIRA is the resource provider, which saves issues in JSON format. Moreover, one issue is regarded as one resource.
- The JIRA adaptor converts the JSON format data to RDF triple format data, with each RDF triple formatted issue tagged with an URI.
- A TRS server lists all the resources available via a JIRA adaptor with their corresponding URIs. JIRA uses WebHook to notify TRS server about changes.
- A TRS client periodically sends a HTTP request to the TRS server to check for updates, then synchronizes with the TRS server.
- The TRS client sends requests with each URI to obtain an initial copy or update existing data and saves the response in the RDF triple format data in Fuseki (i.e., a copy of everything received via the JIRA adaptor).

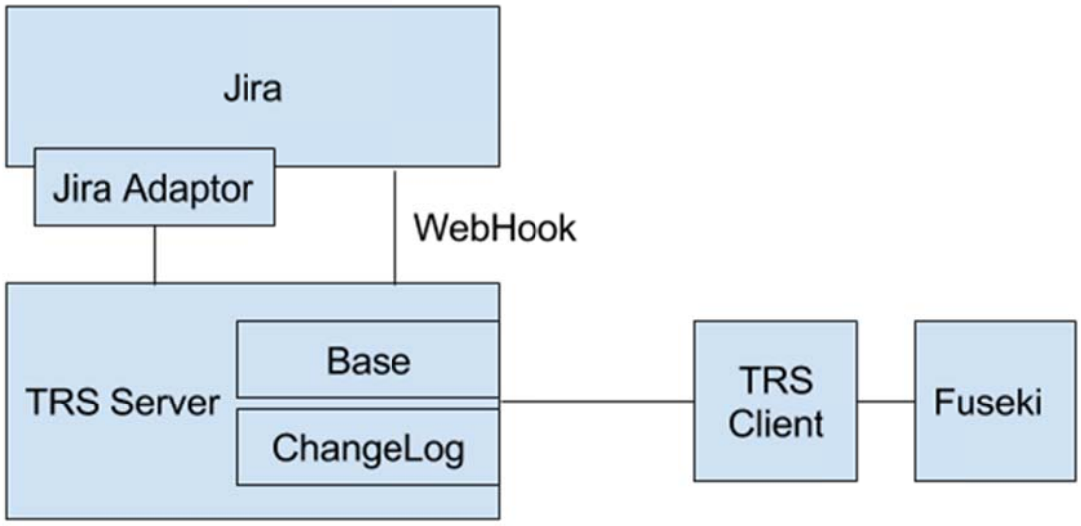

Figure 2-4: JIRA-TRS Architecture

3 A TRS provider for JIRA

This chapter introduces as an example application a TRS provider designed and developed using JIRA. The chapter starts with a description of the installation of JIRA. This is followed by a description of the design and development of a TRS server, Base, and ChangeLog. Finally, the chapter ends with a description of the WebHooks configuration and functional testing of the TRS server and a previously developed client (see Section 2.5.2).

3.1 Installing and configuring JIRA

JIRA can be installed as an application [26] or run via a software development kit (SDK) [27]. This project ran JIRA via the SDK with the JIRA Dashboard accessed via the URI: localhost:2990/jira.

Projects and Issues can be created via the JIRA Dashboard. This process began by creating a sample project and naming it “TRS”. After JIRA was configured, the JIRA Adaptor was run.

3.2 TRS server

This project used the Java API for RESTful Web Services (JAX-RS) [28] to create the TRS web service. TRS server has a root path “/trs”. When a HTTP request is sent to this root path, the TRS server responses with two kinds of replies:

1. The first response type is {“text/plain", "text/html"} and normally returns a String "Hello TRS service." when using a web browser to access this root path.
2. The second response type is “OslcMediaType” [29], which includes Turtle, RDF_XML, XML, JSON, etc. This response type is used to reply to requests from a TRS client. In this response, a TrackedResourceSet object[30] will be sent to the TRS client containing the Base and ChangeLog.

The server knows whether a client or web browser is making a request based upon the request’s header which identified the source of the request.

3.2.1 Base

The first step to create the Base is to use a HTTP GET of issue information from JIRA using the JIRA REST API. The type of the response is JSON. The API is:

http://localhost:2990/jira/rest/api/latest/search

One JQL query result is limited to at most 1000 results per page. There are two simple ways to retrieve more query results:

1. The first way is to use a loop with JQL*:

   http://localhost:2990/jira/rest/api/latest/search?startAt=0&maxResults=50

   This will set the starting point of the search and the number of results returned by each query.

2. The second way is to modify the backend configuration file JIRA-config.properties [17], but this method is not recommended with this project.

The next step is to extract the URI of each issue and then save these URIs in a List.

A Base should have following five properties [31]:

1. Current Base page URI
2. LDP container URI
3. List of members
4. CutoffEvent
5. Next Base page URI

An example JIRA-TRS Base is shown in Figure 3-1.

```
▼<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" xmlns:dcterms="http://purl.org/dc/terms/"
 xmlns:oslc="http://open-services.net/ns/core#" xmlns:ldp="http://www.w3.org/ns/ldp#" xmlns:trs="http://open-
 services.net/ns/core/trs#" xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#" xmlns:xsd="http://www.w3.org/2001/XMLSchema#">
  ▼<ldp:Page rdf:about="http://localhost:8080/Jira-TRS/services/trs/base/1">
    ▼<ldp:pageOf>
      ▼<ldp:Container rdf:about="http://localhost:8080/Jira-TRS/services/trs/base">
         <trs:cutoffEvent rdf:resource="urn:urn-3:cm1.example.com:2017-06-07T08:58:20.000+0200:3"/>
         <rdfs:member rdf:resource="http://localhost:8082/JiraAdaptor/services/serviceProviders/1/resources/issues/10201"/>
         <rdfs:member rdf:resource="http://localhost:8082/JiraAdaptor/services/serviceProviders/1/resources/issues/10202"/>
         <rdfs:member rdf:resource="http://localhost:8082/JiraAdaptor/services/serviceProviders/1/resources/issues/10203"/>
       </ldp:Container>
     </ldp:pageOf>
     <ldp:nextPage rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#nil"/>
   </ldp:Page>
 </rdf:RDF>
```

Figure 3-1: JIRA-TRS Base

According to the TRS protocol [4], a Base may be broken into multiple pages; for example, 30 members per page. In this case, one creates several Bases and saves them in a HashMap, then uses the page number as the key.

All Base pages are required to have a “next page reference”. If a Base page is the last page, it uses “rdf:nil” as the next page reference. An example of a multiple Base page is shown in Figure 3-2.

* Note that we could not utilize a next link in the response, as JQL does not have this functionality.

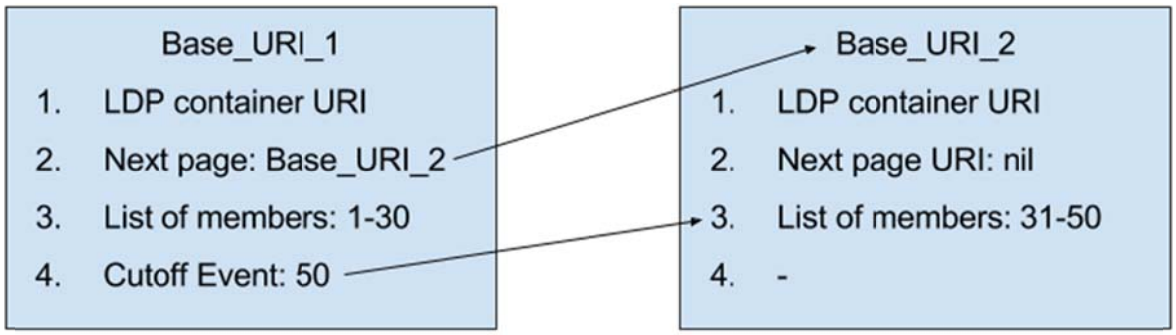

Figure 3-2: Multiple Base

Figure 3-2 shows a Base with two pages: Base_URI_1 and Base_URI_2. Page 1 has a next page reference that points to Page 2. Page 2 is the last page, so the next page reference is "nil". There are totally 50 members saved in the Base: the first 30 members are saved in Page 1 and the remaining 20 members are saved in Page 2, and Page 1 has a CutoffEvent which is the URI of the 50th member.

The Base should regenerate after a certain time, for example, once per week. It is no necessary to regenerate Base if there is no new ChangeEvent during this time interval.

3.2.2 ChangeEvent and ChangeLog

Each change that happens in the TRS provider corresponds to a ChangeEvent. The ChangeLog is a list of all ChangeEvents.

Using WebHook, JIRA sends a JSON-formatted notification to the TRS provider when an issue has changed in JIRA. This notification includes information about the issue such as id, project, creator, created time, and description. If an issue is deleted in JIRA, then the Webhook notification will not contain the deletion time, hence the TRS provider should use its local time.

The ChangeEvent is generated based upon the notification. Each ChangeEvent has the following four properties [32]:

1. ChangeEvent type: creation, modification, or deletion
2. ChangeEvent URI
3. ChangeEvent order
4. Changed issue URI

A ChangeLog entry should have the following three properties:

1. Current ChangeLog page URI
2. List of ChangeEvents
3. Previous ChangeLog Page URI

An example of a JIRA-TRS ChangeLog is shown in Figure 3-3.

```
▼<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" xmlns:dcterms="http://purl.org/dc/terms/"
 xmlns:oslc="http://open-services.net/ns/core#" xmlns:ldp="http://www.w3.org/ns/ldp#" xmlns:trs="http://open-
 services.net/ns/core/trs#" xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#" xmlns:xsd="http://www.w3.org/2001/XMLSchema#">
 ▼<trs:ChangeLog rdf:about="http://localhost:8080/Jira-TRS/services/trs/changeLog/1">
    <trs:previous rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#nil"/>
   ▼<trs:change>
    ▼<trs:Creation rdf:about="urn:urn-3:cm1.example.com:2017-06-07T08:47:45.000+0200:1">
       <trs:order rdf:datatype="http://www.w3.org/2001/XMLSchema#int">1</trs:order>
       <trs:changed rdf:resource="http://localhost:8082/JiraAdaptor/services/serviceProviders/1/resources/issues/10200"/>
     </trs:Creation>
   </trs:change>
   ▼<trs:change>
    ▼<trs:Modification rdf:about="urn:urn-3:cm1.example.com:2017-06-07T08:52:17.802+0200:2">
       <trs:order rdf:datatype="http://www.w3.org/2001/XMLSchema#int">2</trs:order>
       <trs:changed rdf:resource="http://localhost:8082/JiraAdaptor/services/serviceProviders/1/resources/issues/10200"/>
     </trs:Modification>
   </trs:change>
   ▼<trs:change>
    ▼<trs:Deletion rdf:about="urn:urn-3:cm1.example.com:2017-06-07T08:52:33.092+0200:3">
       <trs:order rdf:datatype="http://www.w3.org/2001/XMLSchema#int">3</trs:order>
       <trs:changed rdf:resource="http://localhost:8082/JiraAdaptor/services/serviceProviders/1/resources/issues/10200"/>
     </trs:Deletion>
   </trs:change>
 </trs:ChangeLog>
</rdf:RDF>
```

Figure 3-3: JIRA-TRS ChangeLog

The ChangeLog can be very large, hence it is necessary to segment the ChangeLog into several sub-ChangeLogs. In this case, one creates multiple ChangeLogs and saves them in a HashMap, then uses the page number as the key.

All ChangeLog pages are required to have a “previous page reference”. If a ChangeLog page is the last page, it uses “rdf:nil” as the earlier ChangeLog page reference.

Figure 3-4 shows a ChangeLog with two pages: ChangeLog_URI_1 and ChangeLog_URI_2. There are in total 50 ChangeEvents saved in the ChangeLogs: the newest 30 ChangeEvents are saved in Page 1 and the earliest 20 ChangeEvents are saved in Page 2. Page 1 has a previous page reference points to Page 2, Page 2 is the last page, so the previous page reference is “nil”.

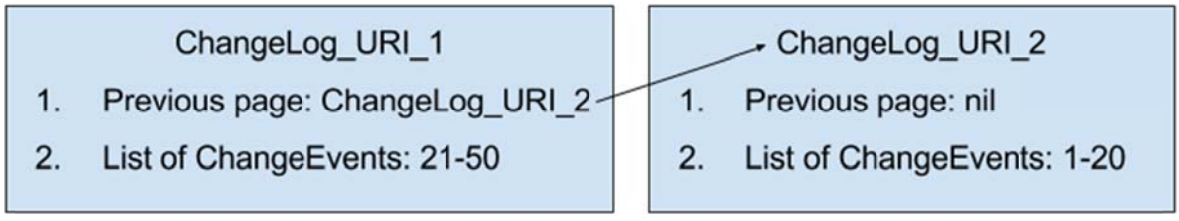

Figure 3-4: Multiple ChangeLog

3.3 Configure WebHooks

Due to a JIRA WebHook limitation, issue modification and issue deletion may send a similar notification to WebHook, making it hard to distinguish a modification WebHook from a deletion WebHook. Therefore, we use three WebHooks instead: creation, modification, and deletion. Table 3-1 shows these WebHooks configurations.

Table 3-1 WebHooks Configuration

creation	http://localhost:8080/jira-trs/services/trs/issues/create
modification	http://localhost:8080/jira-trs/services/trs/issues/modify
deletion	http://localhost:8080/jirs-trs/services/trs/issues/delete

3.4 Run TRS Client

The TRS client will synchronize with TRS server, then access each resource URI to obtain an initial copy and save the copy in Triple Store, or otherwise update the resource according to the ChangeEvent.

3.4.1 Install Fuseki

As described in Section 2.1.4, we have used Apache Fuseki as our triple store. The process of installing this software begins with downloading and extracting Apache Jena Fuseki [33] into a local folder.

After installation, we can start this software by starting a command terminal, going to the Fuseki folder and starting Fuseki with the command “fuseki-server”. The default URI for Fuseki is localhost:3030. We begin by creating a Persistent dataset and naming it “TRS”, as shown in Figure 3-5.

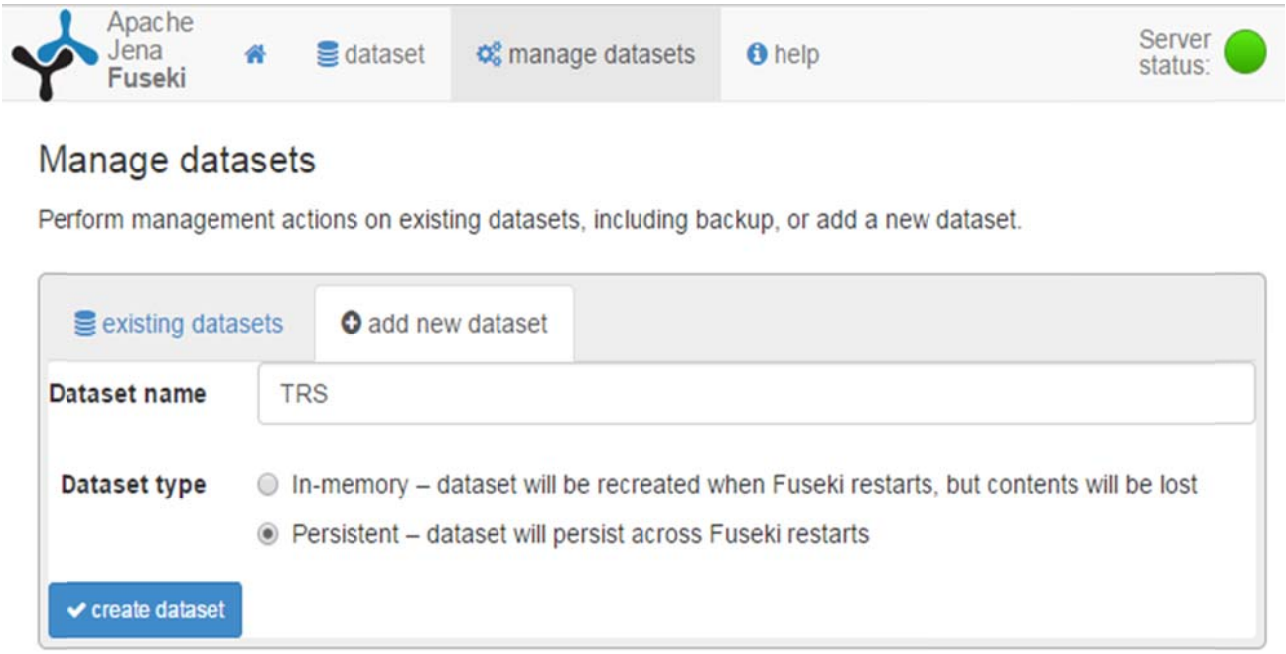

Figure 3-5: Create DataSet in Fuseki

3.4.2 Configure and Run TRS Client

The next step is to setup the TRS server and Triple Store in the TRS client's configuration, and then running the TRS client. The installation and configuration of TRS client is not attached in this thesis, please check Lyo/TRS [34] for the newest version TRS client.

The following SPARQL query can be used to search for data in Fuseki:

```
SELECT ?subject ?predicate ?object
WHERE {
    GRAPH ?g {
            ?subject ?predicate ?object
    }
}
```

An example query result is shown in Figure 3-6.

	subject	predicate	object
1	<http://localhost:8082/JiraAdaptor/services/serviceProviders/1/resources/issues/10201>	<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>	<http://com.scania.jira/jim#Issue>
2	<http://localhost:8082/JiraAdaptor/services/serviceProviders/1/resources/issues/10201>	<http://purl.org/dc/terms/#creator>	<http://localhost:2990/jira/rest/api/2/user?username=admin>
3	<http://localhost:8082/JiraAdaptor/services/serviceProviders/1/resources/issues/10201>	<http://com.scania.jira/jim#project>	<http://localhost:8082/JiraAdaptor/services/serviceProviders/1/resources/projects/10100>
4	<http://localhost:8082/JiraAdaptor/services/serviceProviders/1/resources/issues/10201>	<http://com.scania.jira/jim#id>	"10201"
5	<http://localhost:8082/JiraAdaptor/services/serviceProviders/1/resources/issues/10201>	<http://purl.org/dc/terms/#title>	"test"^^
6	<http://localhost:8082/JiraAdaptor/services/serviceProviders/1/resources/issues/10201>	<http://purl.org/dc/terms/#created>	"2017-06-07T06:58:20Z"^^xsd:dateTime

Figure 3-6: TRS Query result

"An issue with id 10201" is the subject for each of the results shown in Figure 3-6. The predicates are the URIs of each property, while the objects are values of the corresponding property.

Alternatively, we can access the URI http://localhost:3030/TRS/data to browse data in Turtle format. Details about Turtle format can be found in [35].

3.5 Summary

This chapter showed an example of developing a TRS provider for JIRA. Other TRS providers could follow a similar architecture and workflow to build a web service, Base, ChangeEvent, and ChangeLog.

4 Proposal and comparison of potential solutions

This chapter proposes and compares several potential solutions for this project, and then selects the most practical one.

Several metrics were considered while comparing and evaluating the potential solutions:

1. The most significant feature is delay, for the main goal in this project is to reduce delay of the TRS system. Current TRS systems synchronize every 60 seconds, hence it is necessary to decrease the delay to an acceptable value. All of technologies discussed in this chapter can realize a real-time notification system.
2. Another important feature is how the message system is implemented, as some real-time message systems are pull-based. A pull-based message system is incompatible with the design of this project, as we expected to use a push-based architecture.
3. The least important feature, but still interesting feature is extensibility, as in some scenarios it will be necessary to consider QoS or add authentication functions.

4.1 Java Remote Method Invocation (RMI)

Java Remote Method Invocation (RMI) would be a straightforward way to implement a notification function for this project: Using RMI, the TRS server could directly invoke a method in the TRS client and then the TRS client can send a request to the TRS server. However, RMI is not an option in this for project for two reasons:

1) RMI is not really a push technology as the client still needs to send a request to the TRS server.
2) RMI is not-interoperable across containers as transactions are only supported among beans in the same container [36].

4.2 Java Message Service (JMS)

The Java Message Service (JMS) is a Java API which allows applications to send and receive messages. JMS has two message models:

1) Point-to-Point (one to one)

   The JMS P2P model is a traditional PULL-based message model whose structure is shown in Figure 4-1.

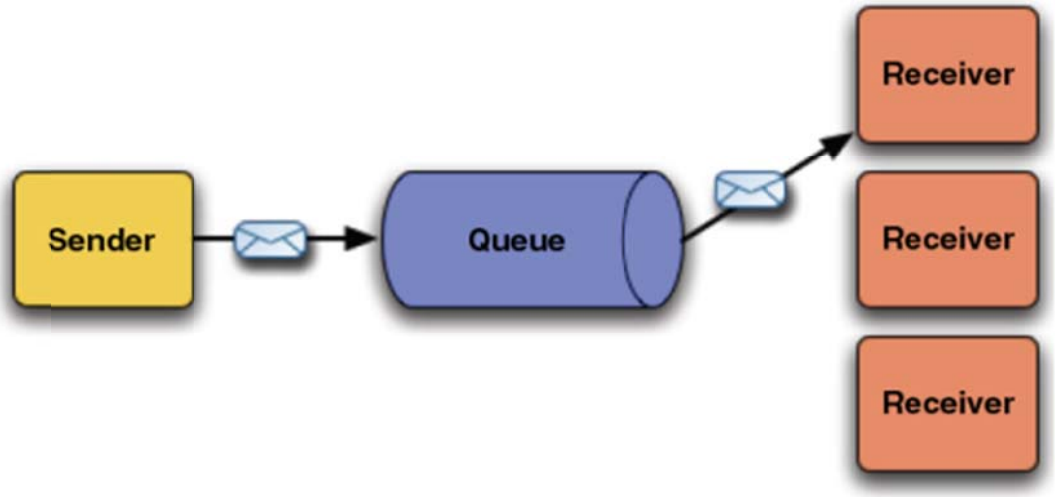

Figure 4-1: JMS Point-to-Point Model [28]

A message with a destination is sent to a queue and then the message is delivered to the destination. The destination replies to the queue with an acknowledgement message when it receives the message; otherwise the queue holds the message until the message is consumed. Finally, the message is removed from the queue.

2) Publish and Subscribe (Pub-Sub) (one to many)

JMS Pub-Sub model is a push-based message model whose structure is shown in Figure 4-2.

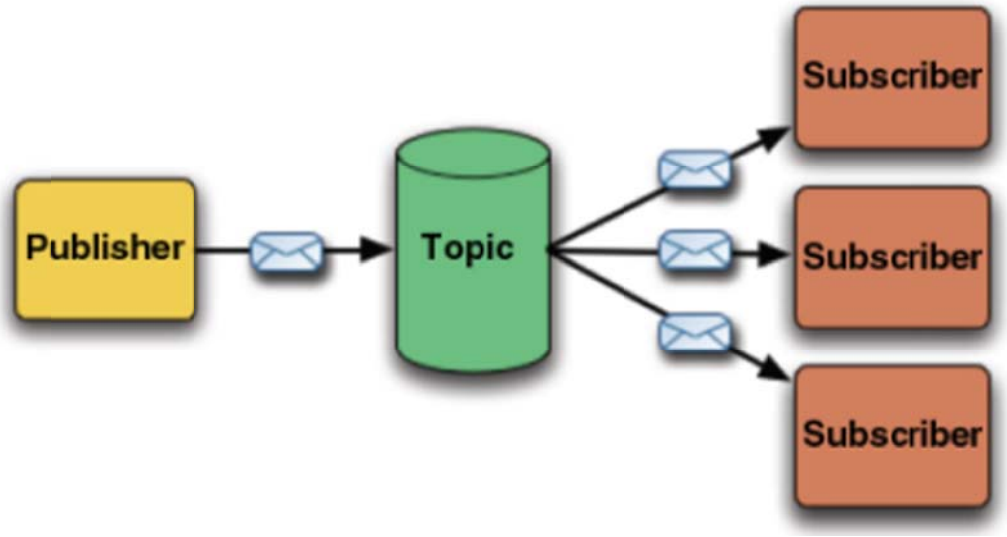

Figure 4-2: JMS Publish-and-Subscribe Model [28]

A message is published to a topic and then all message consumers who have previously subscribed to this topic will receive a copy of the message. In the Pub-Sub model, messages are automatically distributed without pulls or polling.

The JMS Pub-Sub model is a candidate technology for a push-based message architecture. In this project, using this model enables a TRS server to directly push serialized ChangeEvents to TRS clients without requiring either a pull or a polling request from each TRS client.

4.3 Message Queue Telemetry Transport (MQTT)

Message Queue Telemetry Transport (MQTT) is a lightweight Pub-Sub messaging transport [37]. It has a similar structure to the JMS Pub-Sub model. In an MQTT message system, a MQTT broker is responsible for a given topic. A publisher sends a message on this topic to the broker, and then the broker pushes this message to all the subscribers to this topic. MQTT was specifically designed for machine-to-machine (M2M) and Internet of Things (IoT) communication.

MQTT has three levels of QoS. Moreover, it offers guaranteed delivery with QoS levels 1 or 2, which is shown in Table 4-1.

Table 4-1 MQTT QoS level

QoS 0	at most once (unreliable)
QoS 1	at least once
QoS 2	exactly once

The maximum payload size of MQTT is 268,435,455 bytes (i.e., 256 MB) [37], but MQTT has no long-lived storage function nor does it support message fragmentation [38]. According to the requirements for this project (stated in Section 1.4) and the ChangeEvent structure (stated in Section 3.2.2), a MQTT message is sufficient to encapsulate a ChangeEvent.

4.4 Apache Kafka

Apache Kafka is a distributed streaming platform, which provides good scalability [39]. Apache Kafka has a similar structure to a Pub-Sub messaging system, but it is pull-based. Moreover, Apache Kafka provides a log function. A simple example of this logging is shown in Figure 4-3 [39].

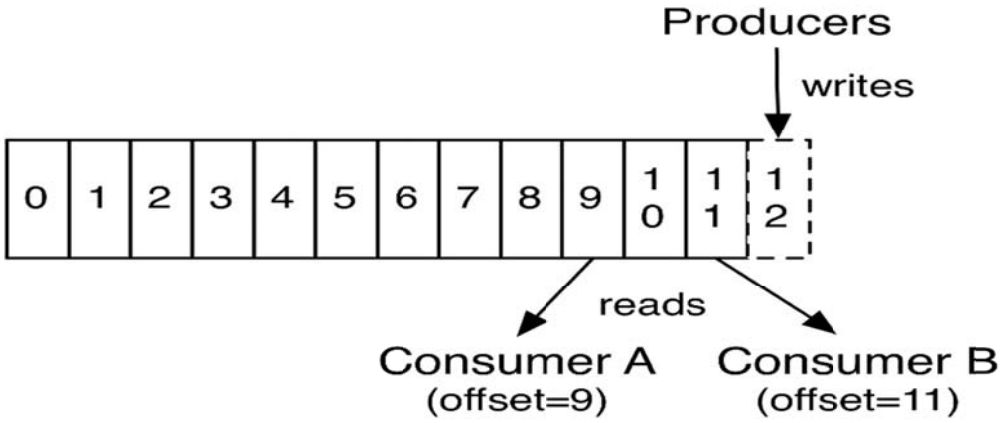

Figure 4-3: Apache Kafka Log

All messages sent to a topic are recorded. A message consumer can read these data starting from an offset. For this project, Apache Kafka could be used to save all ChangeEvents in a log like manner (such as ChangeLog) and then a TRS client can simply read the ChangEvents saved by Apache Kafka. In this case, Apache Kafka would maintain a local ChangeLog. This local copy of the ChangeLog may reduce the load on the TRS server. However, the ChangeLog may be duplicated at each Kafka broker.

4.5 Conclusion

After careful consideration, we decided to use MQTT in this project, for the following reasons:

1) MQTT is push-based, which fulfils the desire for a push-based architecture design.
2) Compared with other candidates, MQTT is lightweight. MQTT has a 2-byte protocol overhead [37], hence the payload-to-overhead ratio is good.
3) One ChangeEvent is a small patch with only has four components (Section 3.2.2) and it is unnecessary to save a duplicated copy of the message in the broker (as would be the case for Apache Kafka).
4) When specifying QoS level 1 or 2, MQTT offers guaranteed delivery. Although this project has not implemented QoS functionality, it provides a potential extension for future work. Theoretically, the MQTT broker will keep a copy of the payload until every client has acknowledged receipt of the payload.
5) MQTT provides simple username/password-based security. The MQTT broker is public, the anonymous users may publish spam to the broker, so it is necessary to only allow the authenticated client to publish message to the topic.

6) Most MQTT implementations are open source. Eclipse Moqsuitto is an open source message broker and Eclipse Paho is an open source MQTT client, hence is easy to modify this code or to embed MQTT into other projects.

Apache Kafka is designed to support large-scale systems, rather than a single customer. Because the broker only needs to forward and push ChangeEvents, it is unnecessary to save any data duplicated in broker. Therefore, Apache Kafka is overkill for this project.

In other situations or with other TRS providers, other message protocols such as Advanced Message Queuing Protocol (AMQP) [40], Streaming Text Oriented Messaging Protocol (STOMP) [41], or even Simple Mail Transfer Protocol (SMTP) [42] might be considered as replacements for MQTT.

In any case, the proposed push functionality should be designed to be a replaceable module, thus using MQTT is simply one possible implementation.

5 A push-based TRS system

This chapter describes the design and implementation of a push-based TRS system. This chapter starts by describing the system's architecture, followed by ChangeEvent serialization and deserialization, and ends with details about the installation and configuration of the MQTT broker, publisher, and subscriber.

5.1 System Architecture

Given the problem statement in Section 1.2, we decided to push the ChangeEvent to the TRS client. After receiving this ChangeEvent, the TRS client updates its data based upon the ChangeEvent without any extra data needing to be sent to the TRS server. The system's architecture is shown in Figure 5-1.

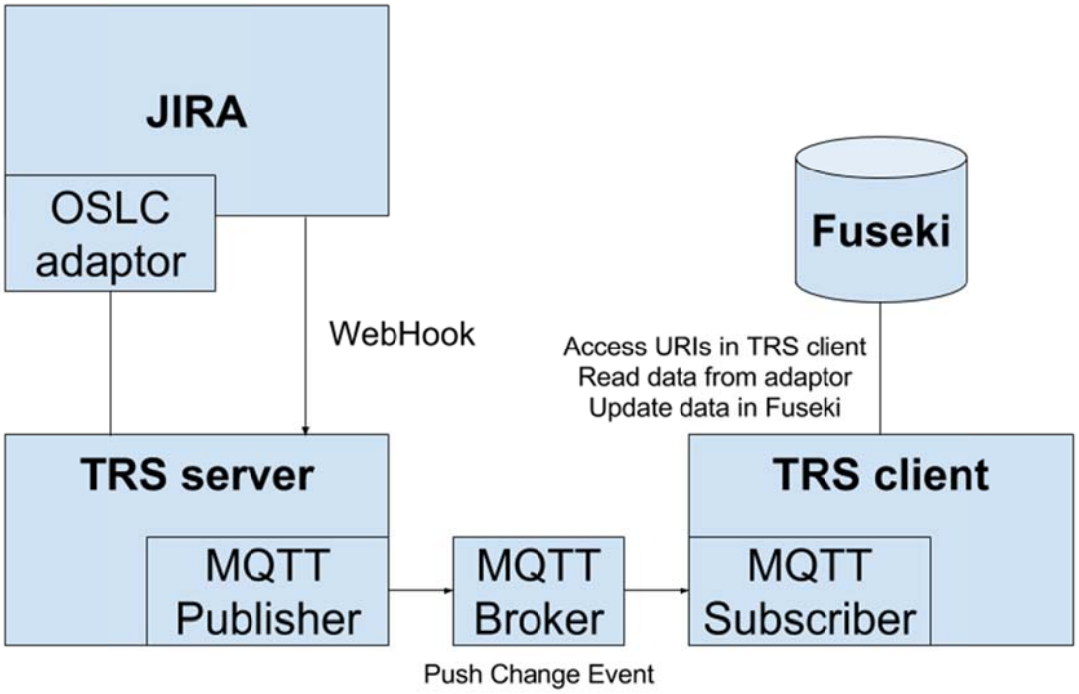

Figure 5-1: Push-based TRS Architecture

After a ChangeEvent is created in the TRS server, the ChangeEvent will be serialized and sent to a specific topic in the MQTT broker. Any TRS client who has subscribed to this topic will receive this serialized ChangeEvent. After receiving a message, the client will deserialize the MQTT message and map the content into a local ChangeEvent. This ChangeEvent will update the TRS client.

5.2 ChangeEvent serialization and deserialization

The ChangeEvent is a Java object, so it must be serialized prior to transmission. The tool used to serialize each ChangeEvent is JenaModelHelper.

5.2.1 Serialization

As described above, ChangeEvent serialization is done by JenaModelHelper. Specifically, we use the method createJenaModel to serialize the ChangeEvent, as shown below:

```
Model changeEventJenaModel =
JenaModelHelper.createJenaModel(new Object[] { changeEvent });
```

This Jena-model ChangeEvent can be converted to a String and encapsulated within a MQTT message, and then sent to the MQTT broker.

5.2.2 Deserialization

Once the TRS client receives the message from the broker, the first step is to extract the payload (i.e., the serialized ChangeEvent). Each ChangeEvent has three properties: "*@trs:order*", "*@trs:changed*", "*@rdf:type*". An example of a serialized ChangeEvent is shown in Figure 5-2.

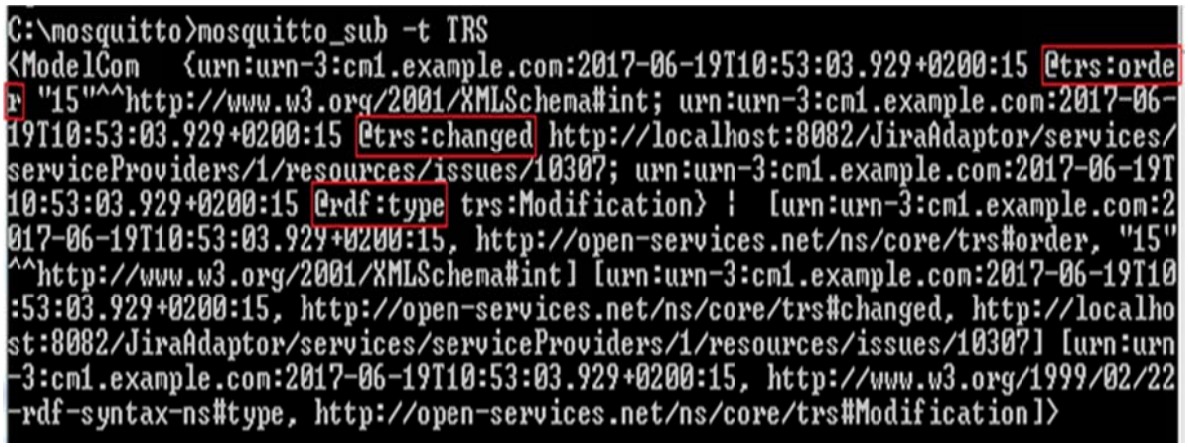

Figure 5-2: ChangeEvent in payload

The TRS client creates a local ChangeEvent with these three properties and updates its local content according to this ChangeEvent.

5.3 MQTT configuration

This project use Mosquitto as the MQTT broker [43] and uses Eclipse Paho [44] as the MQTT publisher and MQTT subscriber. The topic is named "TRS".

Paho is used in both the publisher and subscriber. Paho can easily be embedded and integrated into a Java project. The user only needs to set the MQTT broker address and which topic to publish or subscribe.

This project uses Maven to manage a local repository. Paho can be imported to the repository with:

```
<dependencies>
   <dependency>
      <groupId>org.eclipse.paho</groupId>
      <artifactId>org.eclipse.paho.client.mqttv3</artifactId>
      <version>%VERSION%</version>
   </dependency>
</dependencies>
```

5.3.1 MQTT Broker

Mosquitto is pre-installed in Linux systems. For a Microsoft Windows user, extra libraries need to be included after installation of Mosquitto. The installation of these libraries is described in the file readme-windows.txt that is available after Mosquitto is installed.

5.3.2 MQTT Publisher

The following steps are needed in a program to send a serialized ChangeEvent:

1) Create a MQTT client and connect to TCP port 1883 (this port is reserved for MQTT). This is done with the following code:

```
MqttClient client = new MqttClient("tcp://localhost:1883", "TRSServer");

client.connect();
```

2) Create a MQTT message and set the Jena-model ChangeEvent as the payload with the following code:

```
MqttMessage message = new MqttMessage();

message.setPayload(changeEventJenaModel.toString().getBytes());
```

3) Finally, the application can publish the MQTT message to MQTT broker and disconnects MQTT client with the following code:

```
client.publish(“TRS”, message);

client.disconnect();
```

5.3.3 MQTT Subscriber

For a client to subscribe to and receive published messages from the TRS server, it needs to integrate the following code:

1) Create a MQTT client and set the MqttCallback.

```
MqttClient mqttClient = new MqttClient("tcp://localhost:1883",
"TRSClient");

mqttClient.setCallback(new Listener());
```

2) Connect to the MQTT broker and subscribe the topic.

```
mqttClient.connect();

mqttClient.subscribe("TRS");
```

3) Set MqttCallback

   MqttCallback is an interface, the messageArrived method is called when a message arrives [45]. Next, the message is deserialized and mapped into a local ChangeEvent. This local ChangeEvent must be the same as the original one in the TRS server.

   A method processChangeEvent will be called to process this local ChangeEvent and update the TRS client. In this case, the TRS client is synchronized with TRS server.

5.4 Summary

This chapter presented the design and implementation of a prototype of a push-based TRS architecture where the TRS server pushes a ChangeEvent to the TRS client. The evaluation details on are given in the next chapter.

Normally, the ChangeEvent order should be designed to use continuously increasing sequence numbers. In this case, the TRS client should always check the ChangeEvent to ensure the sequence order of changes are continuous, i.e. without any loss; otherwise, the TRS client should send a normal request to the TRS server to synchronize.

6 Results and Analysis

This chapter evaluates the performance of a JIRA-TRS provider and push-based TRS architecture. We developed a simulator to simulate the operations in JIRA, specifically: issue creation, modification, and deletion. Then we run this simulator while the TRS system is running. In this way, we can monitor the synchronization between TRS Server and TRS client, and calculate cumulative average delay of the synchronization. Additionally, we developed a validator to check if the data in TRS client matches the data in JIRA.

The simulation is run on a PC with the following configuration:

CPU: Intel (R) Core i5-4590 CPU@3.30GHz

Memory (RAM): 8.00GB

Operating System: Windows 7 Enterprise 64-bit

JIRA, OSLC adaptor, TRS server, TRS client, Fuseki, and Mosquitto are running on the same PC. The port settings are shown in Table 6-1.

Table 6-1 Port Setting

Server	Port
JIRA	2990
OSLC adaptor	8082
TRS server	8080
Fuseki	3030
Mosquitto	1883

6.1 Simulation

The simulator can create, modify, and delete issues by sending HTTP requests via the JIRA REST API. The time interval between two operations is randomly selected within a uniform range.

While the push-based TRS system is running, we run the simulator to create 1000 issues in JIRA and the time intervals are uniformly distributed within the range from 1 seconds to 360 seconds*. Then we measure the delay of the synchronization and calculate cumulative moving average delay. The delays of each experiment are shown in Appendix A, while the cumulative moving average delay is shown in Figure 6-1.

* This distribution and range have been selected to purposely vary the system usage and does *not* reflect the expected usage.

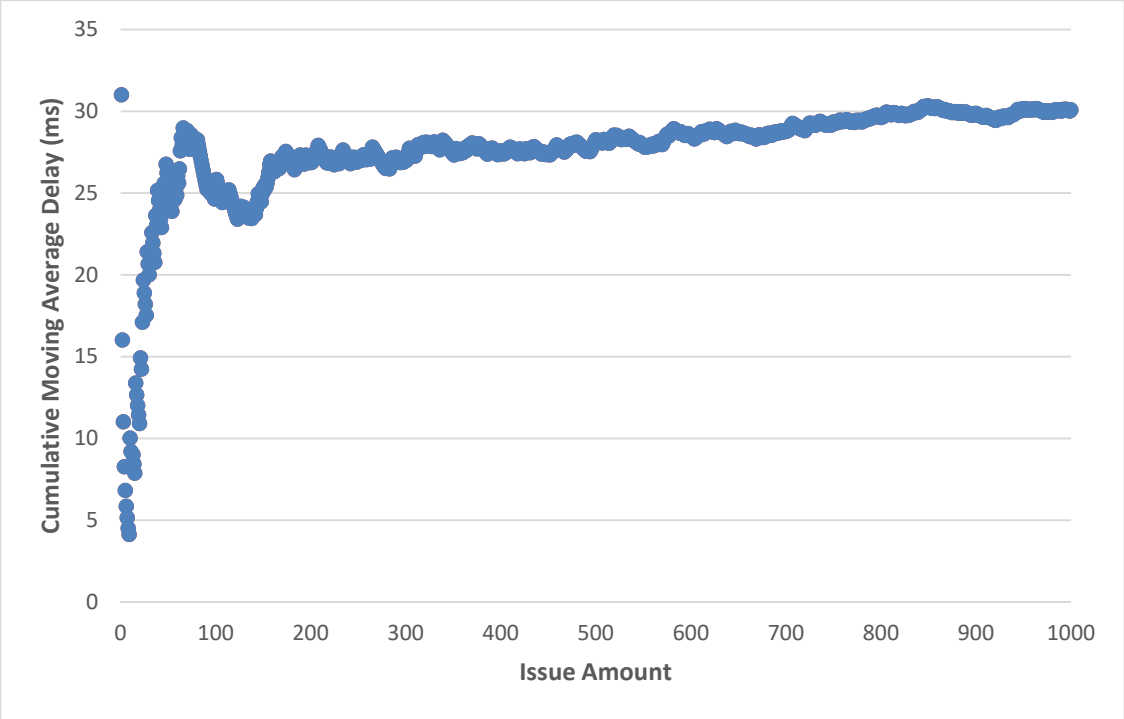

Figure 6-1: Experiment Results

Figure 6-1 shows that after a short period of fluctuation, the cumulative average delay is approaching stable, then it increases with the increasing numbers of issues. The greater memory usage reduces the server's speed. The average delay is stable at around 30 milliseconds. This is a huge improvement compared with the default TRS system that synchronizes only every 60 seconds.

According to the experiment results, the maximum delay is 127 ms and the minimal delay is less than 1 ms. The distribution of delays is plotted in Figure 6-2.

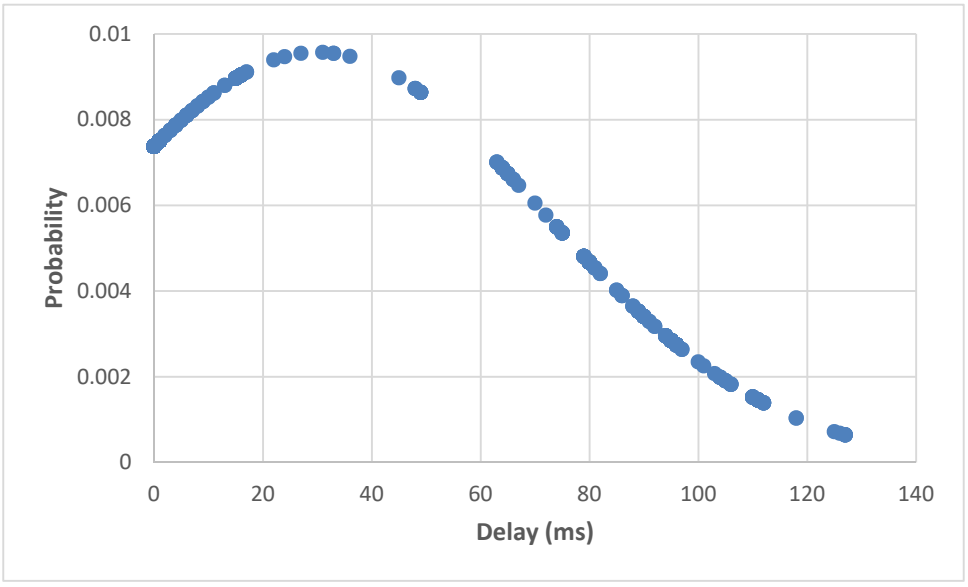

Figure 6-2: Distribution of Delays

6.2 Validation

The validation consists of three steps:

1) Read issues from JIRA in JSON format, then save these issues in HashMap 1.
2) Read issues from Fuseki and convert them to JSON format and save them in HashMap 2.
3) Compare data with the same id between the two HashMaps. Given that all issues in each HashMap use the issue id as their key we can simply iterate through the range of sequence numbers and use them as an id and then compare the JSON objects retrieved from the two HashMaps. If the JSON objects have the same values, then the data matches.

The simulator was run for 40 hours and the time intervals were selected from 1 seconds to 120 seconds, which supposed to execute one operation per minute. The JIRA operations are randomly selected from creation, modification, and deletion. There was a total of 410 issues processed during the simulation.

We validated these 410 issues after the completion of the simulation. We found that all of the data in Fuseki matched the data in JIRA. Thus, there is no data loss during the TRS system synchronization and data transmission.

6.3 Discussion

Based on the simulation and validation, this prototype of push-based TRS architecture fulfils the requirements stated in Section 1.4 on page 3. JIRA-TRS provides JIRA data in RDF format and the TRS client synchronizes with the TRS Server in real-time with the help of MQTT with an average delay of roughly 30 milliseconds.

This project used MQTT as a message system to push serialized ChangeEvents to the TRS client. The TRS client updates its content without requiring any pull requests being sent to the TRS server.

The TRS system no longer uses the pull method for synchronization, the TRS client catches up with TRS server after each change is made in data source. The system delay depends only on server load and network state (hence the delay will increase with network congestion).

This project uses JIRA as the data source. JIRA provides a REST API and WebHook function, which is convenient to implement an OSLC adaptor and push function. When the data source does not provide a REST API or WebHook, the TRS server could use a database listener to monitor changes in the database and generate TRS data.

ChangeEvent serialization is done automatically by JenaModelHelper. While deserializing the MQTT payload and mapping to a local ChangeEvent, the user

needs to modify the deserializer to read correct data format according to the ChangeEvent content (which is shown in Chapter 5.2.2), especially the time format, for the data resource, TRS server, and TRS client may be located in different time zones and may use different time formats, and the TRS client read the time as a String input, therefore it is necessary to convert all timestamps to a common time format and time base, for example, yyyy-mm-dd'T'hh:mm:ss.SSS'Z' [46].

7 Conclusions and Future work

This chapter includes the conclusion of the project and propose the future works.

7.1 Conclusions

This thesis project designed, implemented, and evaluated a prototype push-based TRS architecture and developed a TRS provider for JIRA, which meets the purpose and goals discussed in Chapter 1.

By using a push-based TRS architecture, a TRS client updates its content immediately when changes are made and received via the OSLC adaptor. Thus, reducing server load and reducing network traffic, for there are no synchronization request sent to the server and only the ChangEvent are transmitted via the network.

With the development of JIRA-TRS provider, JIRA may have a TRS adaptor to share data with other tools or software which also follow TRS standard*. For example, JIRA may share data with Bugzilla with the help of TRS.

7.2 Limitations

This project encountered several limitations during the development of the prototype:

1) This push-based TRS architecture is designed for JIRA based upon a case study at Scania, hence it may not support other specific scenarios.

2) The amount of data and experimentation in this project is limited. This push-based TRS architecture needs to be tested with other tools and TRS providers as described in [47].

3) According to the TRS standard [4], the ChangeEvent order may not be a continuous positive integer. As a result, the TRS client cannot know if a ChangeEvent was lost simply by considering the order of ChangeEvents.

4) There was no experiment or investigation of how QoS and network conditions would affect such a push-based TRS system. For actual production use it would be necessary to implement guaranteed MQTT message delivery.

* There are already many tools used within Scania that use TRS. However, information about these tools is not relevant to this thesis.

7.3 Future work

Due to the limited scope of this project and the limitations described above, several things could be (and should be) done in future work, specifically:

1) Consider a specific situation such as a burst of changes happen in JIRA. Due to the network's state, it might be difficult to guarantee that the ChangeEvents arrive at the TRS client in order.

   One possible solution is to check the order of ChangeEvents. If the order of ChangeEvents are not continuous and monotonically increasing, then the TRS client would need to send a normal request to the TRS server. However, as noted earlier this is not in keeping with the TRS protocol, as the order of changes are not required to be continuous.

   Another possible solution is to implement a buffer or waiting time for the TRS server. For example, when a change is made in JIRA, the TRS server might wait 2 seconds and then send the ChangeEvent to TRS client. If another change happens, then the TRS server would refresh this 2 second wait timer, and then send two ChangeEvents in one message. Encapsulating several continuous ChangeEvents in one MQTT message would guarantee ordered ChangeEvents.

2) Add an authentication function to MQTT, as the current implementation is **insecure**. Currently, everyone can publish messages to MQTT broker, thus may cause an error in the TRS client.

3) Implement MQTT's reconnection or payload retransmission function in the TRS client. Given this implementation, test how QoS and network conditions affect the entire system.

4) The ChangEvent order may be not continuous. Therefore, although a given choice of QoS level may provide guaranteed message delivery, the TRS client still does not know if the ChangeEvents are the correct ones. Therefore, it is necessary to investigate how to valid ChangeEvents.

5) Add Java Synchronization or Locks in the TRS server. The current project only considers a single TRS server and client pair. While in other scenarios it would be necessary or desirable to implement a multithreaded and multiuser system. In such a case, it would be necessary to use synchronization mechanisms and locks.

7.4 Required Reflections

This project's prototype push-based TRS architecture realized a greatly reduced delay in comparison with original TRS architecture. This should have an economic and social impact as is speeds up the process or addresses issues that have been found. As the issues being tracking in this industrial setting concern vehicles, some

of these issues can be safety critical, hence facilitating these issues being addressed can have a major impact on society.

Any company, researcher, or academic who wants to share data or integrate software via a protocol, Linked-Data based OSLC specification could be a potential user of the results of this project. Additionally, as an open source project, OSLC needs continuous work and contributions to maintain and promote it. To expand the implementation of Linked Data and OSLC, additional OSLC adaptors and TRS providers for other tools need to be developed.

The implementation of TRS server for JIRA is an example of developing TRS provider and an extension of the OSLC specification. This design and implementation should encourage others to create TRS providers.

As a research project concerning OSLC [25], this project improves the performance of OSLC tool-chain integration and data sharing. Moreover, such a push-based TRS architecture could be part of the next-version TRS protocol.

References

[1] F. Loiret, “ASSUME: Affordable Safe & Secure Mobility Evolution,” *KTH | ASSUME*, 17-Jun-2016. [Online]. Available: https://www.kth.se/en/itm/inst/mmk/forskning/forskningsenheter/mekatronik/modellbaserad-metodik/assume-1.596730.

[2] T. Berners-Lee, “Linked Data - Design Issues,” 18-Jun-2009. [Online]. Available: https://www.w3.org/DesignIssues/LinkedData.html.

[3] OSLC, “What is OSLC?,” *OSLC*. [Online]. Available: http://open-services.net/resources/tutorials/oslc-primer/what-is-oslc/.

[4] S. Speicher, F. Budinsky, and V. Garg, “TrackedResourceSet 2.0,” *OSLC*, 28-Jan-2015. [Online]. Available: http://open-services.net/wiki/core/TrackedResourceSet-2.0/.

[5] bugzilla.org contributors, “What is Bugzilla?,” *Bugzilla*, 27-Feb-2015. [Online]. Available: https://www.bugzilla.org/about/.

[6] OSLC, “Running the example applications - Integrating products with OSLC - Open Services for Lifecycle Collaboration,” *OSLC*. [Online]. Available: http://open-services.net/resources/tutorials/integrating-products-with-oslc/running-the-examples/.

[7] OASIS, “[OSLCCORE-72] TRS should support push-based notification,” *OASIS Technical Committees Issue Tracker*, 04/Aug/16 2:55 PM. [Online]. Available: https://issues.oasis-open.org/browse/OSLCCORE-72?jql=project%20%3D%20OSLCCORE%20AND%20resolution%20%3D%20Unresolved%20AND%20issuetype%20%3D%20Improvement%20ORDER%20BY%20priority%20DESC.

[8] J. Munir, “Information Integration Using a Linked Data Approach,” Master's thesis, KTH, School of Information and Communication Technology (ICT), TRITA-ICT-EX/2015:243, 2015.

[9] OASIS, “Meetings/Telecon2016.06.09,” *OASIS*, 09-Jun-2016. [Online]. Available: https://wiki.oasis-open.org/oslc-core/Meetings/Telecon2016.06.09.

[10] OASIS, “[OSLCCORE-86] Provide TRS consumers with an efficient means of accessing tracked resources,” *OASIS Technical Committees Issue Tracker*, 02/Feb/17 6:44 PM. [Online]. Available: https://issues.oasis-open.org/browse/OSLCCORE-86.

[11] B. McBride, G. Klyne, and J. J. Carroll, “Resource Description Framework (RDF): Concepts and Abstract Syntax,” *W3C*, 10-Feb-2004. [Online]. Available: https://www.w3.org/TR/2004/REC-rdf-concepts-20040210/.

[12] Apache Software Foundation, “Fuseki: serving RDF data over HTTP,” *Apache Jena*. [Online]. Available: https://jena.apache.org/documentation/serving_data/.

[13] E. Prud’hommeaux and A. Seaborne, “SPARQL Query Language for RDF,” *W3C*, 15-Jan-2008. [Online]. Available: https://www.w3.org/TR/rdf-sparql-query/.

[14] ECMA, “JSON,” *Ecma International*, Oct-2013. [Online]. Available: http://www.json.org/.

[15] Atlassian, “JIRA Software - Features,” *Atlassian Software*. [Online]. Available: https://www.atlassian.com/software/jira/features.

[16] Atlassian, “JIRA REST API Version 2 Tutorial,” *Atlassian Developers*. [Online]. Available: https://developer.atlassian.com/jiradev/jira-apis/jira-rest-apis/jira-rest-api-tutorials/jira-rest-api-version-2-tutorial.
[17] Atlassian, “Search JIRA like a boss with JQL,” *Atlassian Documentation*. [Online]. Available: https://confluence.atlassian.com/jiracore/blog/2015/07/search-jira-like-a-boss-with-jql.
[18] T. Davis, “What is a WebHook?,” *Web Hooks*. [Online]. Available: https://webhooks.pbworks.com/w/page/13385124/FrontPage.
[19] Atlassian, “Webhooks,” *Atlassian Developers*. [Online]. Available: https://developer.atlassian.com/jiradev/jira-apis/webhooks.
[20] S. Speicher, “Open Services for Lifecycle Collaboration Change Management Specification Version 2.0,” *OSLC*, 03-Oct-2013. [Online]. Available: http://open-services.net/bin/view/Main/CmSpecificationV2.
[21] IBM, “Indexing your data with the Lifecycle Query Engine (LQE),” *IBM Knowledge Center*. [Online]. Available: https://www.ibm.com/support/knowledgecenter/en/SS2L6K_4.0.5/com.ibm.team.jis.lqe.doc/topics/c_lqe.html.
[22] “Jad El-Khoury,” *KTH*. [Online]. Available: https://www.kth.se/en/itm/inst/mmk/medarbetare/medarbetare-mda/jad-el-khoury-1.57926. [Accessed: 06-Aug-2017].
[23] “Andrii Berezovskyi,” *KTH*. [Online]. Available: https://www.kth.se/en/itm/inst/mmk/medarbetare/medarbetare-mda/andrii-berezovskyi-1.639290.
[24] OSLC, “JenaModelHelper (oslc4j-core-build 1.0 API),” *Eclipse*. [Online]. Available: http://download.eclipse.org/lyo/docs/1.0/apidocs/org/eclipse/lyo/oslc4j/provider/jena/JenaModelHelper.html.
[25] OSLC, “Eclipse Lyo - Enabling Tool Integration with OSLC,” *Eclipse Lyo*. [Online]. Available: http://www.eclipse.org/lyo/.
[26] Atlassian, “Installing JIRA applications on Windows - Atlassian Documentation,” *Atlassian Documentation*. [Online]. Available: https://confluence.atlassian.com/adminjiraserver073/installing-jira-applications-on-windows-861253024.html.
[27] Atlassian, “Getting Started,” *Atlassian Developers*, 04-Dec-2015. [Online]. Available: https://developer.atlassian.com/docs/getting-started.
[28] O. Oracle, “What Are RESTful Web Services?,” *Java Platform, Enterprise Edition: The Java EE Tutorial*. [Online]. Available: https://docs.oracle.com/javaee/7/tutorial/jaxrs001.htm#GIJQY.
[29] OSLC, “OslcMediaType (OSLC4J Build 2.1.2 API),” *Eclipse*. [Online]. Available: http://download.eclipse.org/lyo/docs/core/2.1.2/org/eclipse/lyo/oslc4j/core/model/OslcMediaType.html.
[30] IBM, “TrackedResourceSet (OSLC4J Build 2.1.2 API),” *Eclipse*. [Online]. Available: http://download.eclipse.org/lyo/docs/core/2.1.2/org/eclipse/lyo/core/trs/TrackedResourceSet.html.

[31] IBM, “Base (OSLC4J Build 2.1.2 API),” *Eclipse*. [Online]. Available: http://download.eclipse.org/lyo/docs/core/2.1.2/org/eclipse/lyo/core/trs/Base.html.
[32] IBM, “ChangeLog (OSLC4J Build 2.1.2 API),” *Eclipse*. [Online]. Available: http://download.eclipse.org/lyo/docs/core/2.1.2/org/eclipse/lyo/core/trs/ChangeLog.html.
[33] Apache Software Foundation, “Apache Jena Fuseki,” *Apache Jena*. [Online]. Available: https://jena.apache.org/documentation/fuseki2/.
[34] Eclipse Foundation, “Lyo/TRSReferenceApplication.” [Online]. Available: https://wiki.eclipse.org/Lyo/TRSReferenceApplication.
[35] D. Beckett, T. Berners-Lee, E. Prud’hommeaux, and G. Carothers, “RDF 1.1 Turtle,” *W3C*, 25-Feb-2014. [Online]. Available: https://www.w3.org/TR/turtle/.
[36] Oracle, “J2EE Interoperability,” *Oracle*. [Online]. Available: https://docs.oracle.com/cd/B14099_19/web.1012/b14012/interop.htm.
[37] OASIS, “MQTT Version 3.1.1,” *OASIS*, 29-Oct-2014. [Online]. Available: http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html.
[38] C. Fernando, “Comparison of asynchronous messaging technologies with JMS, AMQP and MQTT,” *Medium*, 12-Mar-2016. [Online]. Available: https://medium.com/@chanakaudaya/comparison-of-asynchronous-messaging-technologies-with-jms-amqp-and-mqtt-4f1bc21c26c5.
[39] Apache Software Foundation, “Apache Kafka: A Distributed Streaming Platform.,” *Apache Kafka*. [Online]. Available: https://kafka.apache.org/intro.html.
[40] OASIS, “AMQP: Advanced Message Queuing Protocol.” [Online]. Available: https://www.amqp.org/.
[41] “STOMP: The Simple Text Oriented Messaging Protocol,” *STOMP*. [Online]. Available: https://stomp.github.io/.
[42] P. Hoffman, “SMTP Service Extension for Secure SMTP over Transport Layer Security,” Feb-2002. [Online]. Available: https://tools.ietf.org/html/rfc3207.
[43] R. A. Light, “Mosquitto: server and client implementation of the MQTT protocol,” *J. Open Source Softw.*, vol. 2, no. 13, May 2017.
[44] Eclipse Foundation, “Eclipse Paho Java Client,” *Eclipse Paho*. [Online]. Available: https://eclipse.org/paho/clients/java/#.
[45] Eclipse Foundation, “MqttCallback,” *Eclipse*. [Online]. Available: https://www.eclipse.org/paho/files/javadoc/org/eclipse/paho/client/mqttv3/MqttCallback.html.
[46] Oracle, “DateTimeFormatter,” *Oracle*. [Online]. Available: https://docs.oracle.com/javase/8/docs/api/java/time/format/DateTimeFormatter.html.
[47] OSLC, “Integrating products with OSLC - Open Services for Lifecycle Collaboration,” *OSLC*. [Online]. Available: http://open-services.net/resources/tutorials/integrating-products-with-oslc/.

Appendix A: Experimental Results

Issue Amount	Delay (ms)	Cumulative Moving Average Delay (ms)	Issue Amount	Delay (ms)	Cumulative Moving Average Delay (ms)	Issue Amount	Delay (ms)	Cumulative Moving Average Delay (ms)
1	31	31.000	46	80	25.586	91	1	25.208
2	1	16.000	47	16	25.382	92	75	25.750
3	1	11.000	48	91	26.750	93	1	25.483
4	0	8.250	49	1	26.224	94	1	25.223
5	1	6.800	50	1	25.720	95	1	24.968
6	1	5.833	51	1	25.235	96	49	25.218
7	1	5.142	52	1	24.769	97	1	24.969
8	0	4.500	53	0	24.301	98	16	24.877
9	1	4.111	54	1	23.870	99	0	24.626
10	63	10.000	55	111	25.454	100	90	25.280
11	1	9.181	56	1	25.017	101	80	25.821
12	0	8.416	57	1	24.596	102	1	25.578
13	16	9.000	58	64	25.275	103	0	25.330
14	1	8.428	59	1	24.864	104	1	25.096
15	0	7.866	60	95	26.033	105	1	24.866
16	96	13.375	61	0	25.606	106	1	24.641
17	1	12.647	62	79	26.467	107	0	24.411
18	1	12.000	63	96	27.571	108	95	25.064
19	1	11.421	64	79	28.375	109	0	24.834
20	1	10.900	65	1	27.953	110	0	24.609
21	95	14.904	66	95	28.969	111	82	25.126
22	0	14.227	67	0	28.537	112	0	24.901
23	80	17.086	68	1	28.132	113	1	24.690
24	79	19.666	69	1	27.739	114	81	25.184
25	0	18.880	70	103	28.814	115	1	24.973
26	1	18.192	71	1	28.422	116	1	24.767
27	0	17.518	72	1	28.041	117	0	24.555
28	126	21.392	73	0	27.657	118	1	24.355
29	0	20.655	74	95	28.567	119	1	24.159
30	1	20.000	75	1	28.200	120	0	23.958
31	65	21.451	76	15	28.026	121	1	23.768
32	1	20.812	77	1	27.675	122	1	23.581
33	79	22.575	78	80	28.346	123	0	23.390
34	1	21.941	79	0	27.987	124	112	24.104
35	0	21.314	80	1	27.650	125	1	23.920
36	1	20.750	81	75	28.234	126	0	23.730
37	127	23.621	82	0	27.890	127	80	24.173
38	1	23.026	83	1	27.566	128	0	23.984
39	106	25.153	84	0	27.238	129	1	23.806
40	0	24.525	85	1	26.929	130	63	24.107
41	1	23.951	86	1	26.627	131	15	24.038
42	1	23.404	87	1	26.333	132	1	23.863
43	1	22.883	88	1	26.045	133	15	23.796
44	112	24.909	89	0	25.752	134	1	23.626
45	1	24.377	90	1	25.477	135	1	23.459

Issue Amount	Delay (ms)	Cumulative Moving Average Delay (ms)	Issue Amount	Delay (ms)	Cumulative Moving Average Delay (ms)	Issue Amount	Delay (ms)	Cumulative Moving Average Delay (ms)
136	65	23.764	183	1	26.420	230	1	26.791
137	1	23.598	184	79	26.706	231	96	27.090
138	1	23.434	185	112	27.167	232	94	27.379
139	48	23.611	186	1	27.026	233	0	27.261
140	79	24.007	187	1	26.887	234	111	27.619
141	0	23.836	188	16	26.829	235	1	27.506
142	1	23.676	189	118	27.312	236	1	27.394
143	80	24.069	190	1	27.173	237	0	27.278
144	80	24.458	191	0	27.031	238	0	27.163
145	96	24.951	192	1	26.895	239	1	27.054
146	1	24.787	193	0	26.756	240	15	27.004
147	0	24.619	194	96	27.113	241	1	26.896
148	1	24.459	195	64	27.302	242	1	26.789
149	96	24.939	196	0	27.163	243	79	27.004
150	81	25.313	197	6	27.055	244	17	26.963
151	1	25.152	198	1	26.924	245	80	27.179
152	79	25.506	199	11	26.844	246	1	27.073
153	1	25.346	200	48	26.950	247	1	26.967
154	64	25.597	201	8	26.855	248	1	26.862
155	75	25.916	202	111	27.272	249	86	27.100
156	80	26.262	203	6	27.167	250	1	26.996
157	96	26.707	204	1	27.039	251	74	27.183
158	64	26.943	205	80	27.297	252	1	27.079
159	1	26.779	206	80	27.553	253	0	26.972
160	1	26.618	207	1	27.425	254	95	27.240
161	1	26.459	208	127	27.903	255	0	27.133
162	1	26.302	209	1	27.775	256	80	27.339
163	79	26.625	210	1	27.647	257	0	27.233
164	0	26.463	211	1	27.521	258	1	27.131
165	80	26.787	212	1	27.396	259	1	27.030
166	5	26.656	213	1	27.272	260	86	27.257
167	0	26.497	214	0	27.144	261	1	27.157
168	101	26.940	215	13	27.079	262	1	27.057
169	9	26.834	216	1	26.958	263	111	27.376
170	96	27.241	217	1	26.838	264	70	27.537
171	0	27.081	218	72	27.045	265	95	27.792
172	1	26.930	219	1	26.926	266	1	27.691
173	95	27.323	220	1	26.809	267	1	27.591
174	64	27.534	221	110	27.185	268	7	27.514
175	8	27.422	222	1	27.067	269	1	27.416
176	7	27.306	223	1	26.950	270	1	27.318
177	1	27.158	224	1	26.834	271	0	27.217
178	1	27.011	225	1	26.720	272	1	27.121
179	1	26.865	226	111	27.092	273	1	27.025
180	0	26.716	227	1	26.977	274	1	26.930
181	24	26.701	228	36	27.017	275	1	26.836
182	1	26.560	229	1	26.903	276	1	26.742

Issue Amount	Delay (ms)	Cumulative Moving Average Delay (ms)	Issue Amount	Delay (ms)	Cumulative Moving Average Delay (ms)	Issue Amount	Delay (ms)	Cumulative Moving Average Delay (ms)
277	1	26.649	324	0	27.925	371	1	27.978
278	0	26.553	325	1	27.843	372	1	27.905
279	11	26.498	326	80	28.003	373	1	27.833
280	80	26.689	327	1	27.920	374	1	27.762
281	0	26.594	328	0	27.835	375	1	27.690
282	16	26.556	329	64	27.945	376	80	27.829
283	1	26.466	330	85	28.118	377	94	28.005
284	95	26.707	331	1	28.036	378	1	27.933
285	80	26.894	332	1	27.954	379	1	27.862
286	96	27.136	333	1	27.873	380	0	27.789
287	1	27.045	334	1	27.793	381	1	27.719
288	2	26.958	335	1	27.713	382	1	27.649
289	9	26.896	336	0	27.630	383	1	27.579
290	110	27.182	337	92	27.821	384	1	27.510
291	8	27.116	338	95	28.020	385	0	27.438
292	1	27.027	339	95	28.218	386	1	27.370
293	1	26.938	340	0	28.135	387	95	27.545
294	3	26.857	341	1	28.055	388	90	27.706
295	27	26.857	342	16	28.020	389	1	27.637
296	65	26.986	343	0	27.938	390	1	27.569
297	1	26.898	344	1	27.860	391	94	27.739
298	22	26.882	345	1	27.782	392	1	27.670
299	111	27.163	346	0	27.702	393	1	27.603
300	1	27.076	347	1	27.625	394	1	27.535
301	1	26.990	348	1	27.548	395	1	27.468
302	90	27.198	349	0	27.469	396	1	27.401
303	112	27.478	350	0	27.391	397	1	27.335
304	104	27.730	351	1	27.316	398	80	27.467
305	1	27.642	352	90	27.494	399	1	27.401
306	3	27.562	353	104	27.711	400	96	27.572
307	5	27.488	354	1	27.635	401	1	27.506
308	6	27.418	355	1	27.560	402	0	27.437
309	1	27.333	356	1	27.485	403	1	27.372
310	1	27.248	357	0	27.408	404	80	27.502
311	111	27.517	358	96	27.600	405	1	27.437
312	96	27.737	359	1	27.526	406	95	27.603
313	95	27.952	360	0	27.450	407	1	27.538
314	1	27.866	361	65	27.554	408	96	27.705
315	1	27.780	362	95	27.740	409	1	27.640
316	95	27.993	363	1	27.666	410	90	27.792
317	0	27.905	364	0	27.590	411	1	27.727
318	1	27.820	365	111	27.819	412	33	27.740
319	111	28.081	366	0	27.743	413	1	27.675
320	1	27.996	367	95	27.926	414	17	27.649
321	0	27.909	368	1	27.853	415	0	27.583
322	88	28.096	369	33	27.867	416	1	27.519
323	1	28.012	370	96	28.051	417	1	27.455

Issue Amount	Delay (ms)	Cumulative Moving Average Delay (ms)	Issue Amount	Delay (ms)	Cumulative Moving Average Delay (ms)	Issue Amount	Delay (ms)	Cumulative Moving Average Delay (ms)
418	1	27.392	465	1	27.578	512	1	28.138
419	75	27.505	466	15	27.551	513	1	28.085
420	105	27.690	467	0	27.492	514	0	28.031
421	10	27.648	468	111	27.670	515	80	28.132
422	1	27.585	469	0	27.611	516	95	28.261
423	1	27.522	470	80	27.723	517	1	28.208
424	1	27.459	471	90	27.855	518	95	28.337
425	1	27.397	472	1	27.798	519	127	28.527
426	94	27.553	473	111	27.974	520	1	28.475
427	91	27.702	474	1	27.917	521	1	28.422
428	0	27.637	475	74	28.014	522	80	28.521
429	1	27.575	476	0	27.955	523	0	28.466
430	1	27.513	477	1	27.899	524	1	28.414
431	0	27.450	478	106	28.062	525	0	28.360
432	105	27.629	479	1	28.006	526	1	28.307
433	89	27.771	480	75	28.104	527	0	28.254
434	1	27.709	481	1	28.047	528	96	28.382
435	80	27.829	482	1	27.991	529	1	28.330
436	1	27.768	483	0	27.933	530	1	28.279
437	1	27.707	484	1	27.878	531	49	28.318
438	1	27.646	485	1	27.822	532	0	28.265
439	1	27.585	486	1	27.767	533	95	28.390
440	0	27.522	487	1	27.712	534	0	28.337
441	15	27.494	488	1	27.657	535	96	28.463
442	0	27.432	489	0	27.601	536	0	28.410
443	1	27.372	490	1	27.546	537	0	28.357
444	95	27.524	491	95	27.684	538	1	28.306
445	1	27.465	492	1	27.630	539	0	28.254
446	0	27.403	493	1	27.576	540	0	28.201
447	0	27.342	494	0	27.520	541	15	28.177
448	80	27.459	495	80	27.626	542	1	28.127
449	1	27.400	496	97	27.766	543	1	28.077
450	0	27.340	497	75	27.861	544	6	28.036
451	49	27.388	498	95	27.995	545	1	27.987
452	1	27.329	499	89	28.118	546	80	28.082
453	80	27.445	500	94	28.250	547	0	28.031
454	79	27.559	501	1	28.195	548	1	27.981
455	1	27.501	502	1	28.141	549	1	27.932
456	65	27.583	503	1	28.087	550	1	27.883
457	66	27.667	504	90	28.210	551	1	27.834
458	96	27.816	505	1	28.156	552	1	27.786
459	80	27.930	506	0	28.100	553	66	27.855
460	1	27.871	507	1	28.047	554	0	27.805
461	1	27.813	508	89	28.167	555	64	27.870
462	0	27.753	509	80	28.269	556	1	27.821
463	1	27.695	510	1	28.215	557	65	27.888
464	0	27.635	511	16	28.191	558	64	27.953

Issue Amount	Delay (ms)	Cumulative Moving Average Delay (ms)	Issue Amount	Delay (ms)	Cumulative Moving Average Delay (ms)	Issue Amount	Delay (ms)	Cumulative Moving Average Delay (ms)
559	1	27.905	606	0	28.389	653	95	28.719
560	1	27.857	607	80	28.474	654	6	28.685
561	111	28.005	608	80	28.559	655	15	28.664
562	0	27.955	609	1	28.513	656	1	28.621
563	45	27.985	610	79	28.596	657	0	28.578
564	1	27.937	611	111	28.731	658	49	28.609
565	80	28.030	612	0	28.684	659	1	28.567
566	92	28.143	613	1	28.639	660	1	28.525
567	0	28.093	614	1	28.594	661	0	28.482
568	1	28.045	615	96	28.704	662	1	28.441
569	1	27.998	616	82	28.790	663	96	28.542
570	1	27.950	617	1	28.745	664	0	28.500
571	111	28.096	618	80	28.828	665	1	28.458
572	96	28.215	619	0	28.781	666	0	28.415
573	95	28.331	620	96	28.890	667	1	28.374
574	118	28.487	621	1	28.845	668	1	28.333
575	80	28.577	622	1	28.800	669	1	28.292
576	1	28.529	623	0	28.754	670	96	28.394
577	1	28.481	624	0	28.708	671	110	28.515
578	80	28.570	625	16	28.688	672	49	28.546
579	125	28.737	626	110	28.817	673	1	28.505
580	1	28.689	627	95	28.923	674	1	28.464
581	79	28.776	628	1	28.878	675	0	28.422
582	112	28.919	629	1	28.834	676	1	28.381
583	1	28.871	630	1	28.790	677	80	28.457
584	0	28.821	631	1	28.746	678	0	28.415
585	1	28.774	632	4	28.707	679	110	28.536
586	1	28.726	633	1	28.663	680	80	28.611
587	16	28.705	634	1	28.619	681	1	28.571
588	0	28.656	635	0	28.574	682	80	28.646
589	80	28.743	636	0	28.529	683	0	28.604
590	0	28.694	637	1	28.486	684	1	28.564
591	1	28.648	638	1	28.443	685	1	28.524
592	1	28.601	639	110	28.571	686	80	28.599
593	0	28.553	640	1	28.528	687	96	28.697
594	1	28.506	641	96	28.633	688	0	28.655
595	80	28.593	642	110	28.760	689	1	28.615
596	1	28.546	643	1	28.716	690	111	28.734
597	74	28.623	644	0	28.672	691	1	28.694
598	1	28.576	645	1	28.629	692	1	28.654
599	0	28.529	646	80	28.708	693	86	28.737
600	1	28.483	647	100	28.819	694	1	28.697
601	1	28.437	648	1	28.776	695	95	28.792
602	1	28.392	649	0	28.731	696	0	28.751
603	0	28.344	650	11	28.704	697	1	28.711
604	0	28.298	651	0	28.660	698	80	28.785
605	112	28.436	652	1	28.618	699	1	28.745

Issue Amount	Delay (ms)	Cumulative Moving Average Delay (ms)	Issue Amount	Delay (ms)	Cumulative Moving Average Delay (ms)	Issue Amount	Delay (ms)	Cumulative Moving Average Delay (ms)
700	80	28.818	747	1	29.186	794	80	29.727
701	0	28.777	748	1	29.148	795	0	29.690
702	95	28.871	749	90	29.229	796	81	29.755
703	0	28.830	750	110	29.337	797	4	29.722
704	127	28.970	751	1	29.299	798	0	29.685
705	80	29.042	752	1	29.261	799	0	29.648
706	111	29.158	753	74	29.321	800	0	29.611
707	88	29.241	754	1	29.283	801	89	29.685
708	1	29.201	755	105	29.384	802	1	29.649
709	1	29.162	756	1	29.346	803	96	29.732
710	1	29.122	757	104	29.445	804	104	29.824
711	1	29.082	758	0	29.406	805	80	29.886
712	1	29.043	759	16	29.388	806	75	29.942
713	4	29.008	760	1	29.351	807	6	29.913
714	1	28.969	761	64	29.396	808	1	29.877
715	2	28.931	762	49	29.422	809	1	29.841
716	1	28.892	763	7	29.393	810	0	29.804
717	48	28.919	764	95	29.479	811	1	29.769
718	0	28.878	765	0	29.440	812	110	29.868
719	1	28.840	766	1	29.403	813	1	29.832
720	1	28.801	767	13	29.382	814	95	29.912
721	96	28.894	768	1	29.345	815	1	29.877
722	96	28.987	769	1	29.308	816	1	29.841
723	110	29.099	770	80	29.374	817	0	29.805
724	81	29.171	771	1	29.337	818	0	29.768
725	96	29.263	772	1	29.300	819	1	29.733
726	1	29.224	773	64	29.345	820	95	29.813
727	1	29.185	774	96	29.431	821	1	29.778
728	74	29.247	775	1	29.394	822	97	29.860
729	1	29.208	776	1	29.358	823	1	29.825
730	1	29.169	777	95	29.442	824	1	29.790
731	1	29.131	778	1	29.406	825	0	29.753
732	104	29.233	779	1	29.369	826	0	29.717
733	1	29.195	780	4	29.337	827	89	29.789
734	95	29.284	781	106	29.435	828	0	29.753
735	75	29.346	782	1	29.398	829	15	29.735
736	49	29.373	783	95	29.482	830	105	29.826
737	1	29.335	784	4	29.450	831	1	29.791
738	1	29.296	785	90	29.527	832	88	29.861
739	1	29.258	786	0	29.489	833	1	29.827
740	1	29.220	787	80	29.554	834	90	29.899
741	1	29.182	788	1	29.517	835	95	29.977
742	1	29.144	789	96	29.602	836	1	29.942
743	80	29.212	790	0	29.564	837	0	29.906
744	0	29.173	791	89	29.639	838	74	29.959
745	1	29.135	792	0	29.602	839	0	29.923
746	95	29.223	793	79	29.664	840	105	30.013

Issue Amount	Delay (ms)	Cumulative Moving Average Delay (ms)	Issue Amount	Delay (ms)	Cumulative Moving Average Delay (ms)	Issue Amount	Delay (ms)	Cumulative Moving Average Delay (ms)
841	95	30.090	888	0	29.891	935	111	29.695
842	1	30.055	889	80	29.948	936	95	29.764
843	75	30.109	890	1	29.915	937	0	29.733
844	110	30.203	891	1	29.883	938	89	29.796
845	94	30.279	892	1	29.850	939	1	29.765
846	1	30.244	893	1	29.818	940	80	29.819
847	1	30.210	894	0	29.785	941	64	29.855
848	75	30.262	895	1	29.753	942	111	29.941
849	80	30.321	896	89	29.819	943	90	30.005
850	15	30.303	897	1	29.787	944	111	30.091
851	1	30.269	898	1	29.755	945	1	30.060
852	1	30.234	899	80	29.810	946	0	30.028
853	0	30.199	900	74	29.860	947	9	30.006
854	1	30.165	901	1	29.827	948	96	30.075
855	85	30.229	902	0	29.794	949	96	30.145
856	1	30.195	903	1	29.763	950	1	30.114
857	0	30.159	904	0	29.730	951	0	30.083
858	80	30.217	905	16	29.714	952	95	30.151
859	74	30.268	906	1	29.683	953	1	30.120
860	1	30.234	907	1	29.651	954	0	30.089
861	1	30.200	908	0	29.618	955	1	30.058
862	5	30.171	909	95	29.690	956	75	30.105
863	1	30.137	910	1	29.659	957	1	30.075
864	1	30.104	911	94	29.729	958	64	30.110
865	0	30.069	912	1	29.698	959	1	30.080
866	1	30.035	913	1	29.667	960	79	30.131
867	80	30.093	914	1	29.635	961	1	30.100
868	1	30.059	915	1	29.604	962	16	30.086
869	15	30.042	916	1	29.573	963	64	30.121
870	1	30.009	917	1	29.541	964	66	30.158
871	1	29.975	918	15	29.526	965	1	30.128
872	63	30.013	919	1	29.495	966	1	30.098
873	1	29.980	920	1	29.464	967	1	30.068
874	1	29.947	921	1	29.433	968	1	30.038
875	0	29.913	922	80	29.488	969	0	30.007
876	80	29.970	923	95	29.559	970	1	29.977
877	3	29.939	924	90	29.624	971	1	29.947
878	1	29.906	925	0	29.592	972	1	29.917
879	89	29.973	926	1	29.561	973	127	30.017
880	1	29.940	927	15	29.545	974	16	30.003
881	0	29.906	928	106	29.628	975	1	29.973
882	2	29.875	929	90	29.693	976	0	29.942
883	111	29.967	930	1	29.662	977	96	30.010
884	1	29.934	931	0	29.630	978	4	29.983
885	1	29.901	932	67	29.670	979	0	29.953
886	1	29.869	933	0	29.638	980	0	29.922
887	80	29.925	934	1	29.608	981	94	29.987

Issue Amount	Delay (ms)	Cumulative Moving Average Delay (ms)
982	0	29.957
983	111	30.039
984	80	30.090
985	10	30.070
986	1	30.040
987	74	30.085
988	1	30.055
989	0	30.025
990	1	29.995
991	79	30.045
992	90	30.105
993	1	30.076
994	1	30.047
995	95	30.112
996	1	30.083
997	1	30.054
998	0	30.024
999	15	30.009
1000	96	30.075

TRITA-ICT-EX-2017:138