Linked Data Search and Browse Application

Chao Cai
Fatemeh Shirazi Nasab
Abstract

Linked Data Search and Browse Application

Chao Cai and Fatemeh Shirazi Nasab

In order to achieve the ISO 26262 standard on the perspective of requirements traceability, a huge volume of data has been converted into RDF format, and been stored in a Triple store. The need to have a web application to search and browse that RDF data has been raised. In order to build this application, several open-source components such as Apache Solr, Apache Jena, Fuseki and Technologies such as Java, HTML, CSS, and Javascript have been used. The application has been evaluated with SUS method, an industry standard and reliable tool for measuring the system usability, by six engineers from Scania, the evaluation result in general is positive, five out of six participants successfully retrieved their desired search results.

Keyword: ISO 26262, Traceability, Triple store, Web application
Acknowledgement

We wish to extend our deepest gratitude to our advisors, Dr. Mattias Nyberg and Dr. Tore Risch, for their thoughtful guidance throughout our master thesis. We appreciate many fruitful discussions and helpful advice from the members of the Scania RESA group. We would also like to thank our colleagues in Scania, MSc. Einarson, MSc. Thydén, and Mr. Vattinen, for their support.

We could not have finished this project without the support from our family and friends, especially our friend, Mohamed Redha Redjimi, who has been supporting us throughout this thesis.
Contents

1 Introduction ................................................................................................ 13

2 Background ................................................................................................ 16
  2.1 The ISO 26262 standard ................................................................ 16
  2.2 OSLC .............................................................................................. 16
  2.3 Semantic Web ................................................................................. 17
    2.3.1 The semantic web layer cake .......................................... 17
    2.3.2 Semantic search engines ................................................. 18
  2.4 Linked data ..................................................................................... 19
  2.5 RDF ................................................................................................. 19
    2.5.1 Database for RDF ............................................................ 19
  2.6 Solr .................................................................................................. 20
    2.6.1 Solr Architecture ................................................................ 20
    2.6.2 Searching in Solr ............................................................. 21
    2.6.3 Three configuration files in Solr ..................................... 22
    2.6.4 Indexing RDF data with Solr .......................................... 24
  2.7 Apache Jena .................................................................................... 26
  2.8 UI design for searching and browsing .......................................... 27

3 The Search and Browse System ............................................................... 29
  3.1 User Interface ................................................................................. 29
  3.2 System Implementation ................................................................. 29
    3.2.1 System Architecture ........................................................ 29
    3.2.2 The user interface ............................................................ 31
    3.2.3 Middle component ........................................................... 35
    3.2.4 Backend ............................................................................ 36

4 System Evaluation ..................................................................................... 38
  4.1 Usability Evaluation ....................................................................... 38
  4.2 Comments and Suggestions from Scania Engineers .................... 39

5 Conclusion and future work ...................................................................... 41
  5.1 Future work .................................................................................... 42

References ........................................................................................................ 44

Appendix : Appendix A : SUS-Questionnaire ............................................... 45
List of Tables

Table 4.1: SUS Questionnaire Results ........................................................... 39
List of Figures

Figure 1.1: Espresso 2 ............................................................... 14
Figure 1.2: SesammTool .......................................................... 15
Figure 2.1: Semantic Web Stack ............................................... 18
Figure 2.2: A Triple sample ..................................................... 20
Figure 2.3: Architecture of Solr .............................................. 21
Figure 2.4: An overview of steps involved in a Solr search .......... 22
Figure 2.5: Analyzer, Tokenizer and Filter ............................... 25
Figure 2.6: Jena Assembler .................................................... 27
Figure 3.1: First screen of LD Search & Browse App ............... 30
Figure 3.2: Second screen of LD Search & Browse App .......... 30
Figure 3.3: Third screen of LD Search & Browse App ............. 31
Figure 3.4: System Architecture of LD Search & Browse App .... 32
Figure 3.5: Configuration of Suggestion Component in solrconfig.xml 34
### Abbreviations and Acronyms

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>ASIL</td>
<td>Automotive Safety Integrity Levels</td>
</tr>
<tr>
<td>CSS</td>
<td>Cascading Style Sheets</td>
</tr>
<tr>
<td>DBMS</td>
<td>Database Management System</td>
</tr>
<tr>
<td>HCI</td>
<td>Human-Computer Interaction</td>
</tr>
<tr>
<td>HTML</td>
<td>Hypertext Markup Language</td>
</tr>
<tr>
<td>HTTP</td>
<td>Hyper Text Transfer Protocol</td>
</tr>
<tr>
<td>IDE</td>
<td>Integrated Development Environment</td>
</tr>
<tr>
<td>IDX</td>
<td>Index</td>
</tr>
<tr>
<td>IR</td>
<td>Information Retrieval</td>
</tr>
<tr>
<td>JSON</td>
<td>Java Script Object Notation</td>
</tr>
<tr>
<td>JSON-LD</td>
<td>Java Script Object Notation for Linked Data</td>
</tr>
<tr>
<td>MVC</td>
<td>Model View Controller</td>
</tr>
<tr>
<td>OSLC</td>
<td>Open Service for Lifecycle Collaboration</td>
</tr>
<tr>
<td>OWL</td>
<td>Web Ontology Language</td>
</tr>
<tr>
<td>RDBMS</td>
<td>Relational Database Management System</td>
</tr>
<tr>
<td>RDF</td>
<td>Resource Description Framework</td>
</tr>
<tr>
<td>RDFS</td>
<td>Resource Description Framework Schema</td>
</tr>
<tr>
<td>REST</td>
<td>Representational State Transfer</td>
</tr>
<tr>
<td>SQL</td>
<td>Structured Query Language</td>
</tr>
<tr>
<td>SUS</td>
<td>System Usability Scale</td>
</tr>
<tr>
<td>TDB</td>
<td>Triple Database</td>
</tr>
<tr>
<td>UI</td>
<td>User Interface</td>
</tr>
<tr>
<td>URI</td>
<td>Uniform Resource Identifiers</td>
</tr>
<tr>
<td>WAR</td>
<td>Web Application Archive</td>
</tr>
<tr>
<td>W3C</td>
<td>World Wide Web Consortium</td>
</tr>
<tr>
<td>XML</td>
<td>Extensible Markup Language</td>
</tr>
</tbody>
</table>
1. Introduction

This degree project was carried out between January, 2016 and June, 2016 at Scania CV AB, a Swedish automotive industry manufacturer. Scania is making the transition to ISO 26262, which is a standard about functional safety, and a project called Espresso2 (Figure 1.1) is proposed to solve the requirements’ traceability problem during the embedded system development. In order to solve the traceability problem, data is converted into RDF (Resource Description Framework), which is a data model for representing information about resources in the web. In order to meet the full traceability requirements of the ISO 26262 standard, Scania is building a platform to store all the data that generated during the embedded system development, in this way, every test case will link to its requirement(s), and every requirement will be sure to have test case(s). The meta modeling of the ISO 26262 standard in the structure of RDF can be used for achieving traceability. Since a huge volume of data gathered from different tools is going to be converted to RDF format, the need for an application where engineers can search and browse their interested data is a must. The search applications that engineers are using now for accessing Scania databases are based on relational databases and they are all limited to a single data source. In other words, if engineers would like to access data in different sources they have to switch among tools which are time consuming and inconvenience (Figure 1.2).

The main goal of the project is to make a prototype of a web application for searching and browsing Scania data, and demonstrate the possibility of searching and browsing RDF data. The project was carried out in the process of knowledge preparation, literature studies, UI design, implementation, and usability evaluation.

The application presented in this paper is called "LD Search & Browse App" which is a RESTful (Representational State Transfer) web application for searching and browsing RDF data. REST is a software architecture to define how the components of large distributed system should interact, specially the interaction between clients and server when requesting and passing resources.

This thesis report is composed of five chapters where different perspectives of the project implementation are discussed.

Chapter two presents the background of the project and a review of related
Figure 1.1. Espresso 2 is a project at Scania in order to solve functional safety problem. This image shows a chain of Scania’s tools which are connected to interchange data. This connection is based on OSLC.

technologies. This review includes OSLC (Open Service for Lifecycle Collaboration), semantic web, Linked data, and RDF, which are infrastructures of integrating tools and software. Semantic search engine and user-interface design are also introduced in this chapter.

Chapter three concentrates on the search and browse application for RDF data and implementation of the application. This chapter illustrates the user case scenario and the details of technical implementation.

Chapter four describes a questionnaire to evaluate the usefulness of the implemented prototype. It is about evaluation of the system through a survey from engineers at Scania and their comments about the system.

Chapter five analyzes the results of the evaluation. It expresses the negative and positive responses, and how can the project to be improved. It is also contains a list of tasks that can be done in the future due to improve the project.
Figure 1.2. SesammTool, one of the tools to access data used by engineers at Scania
2. Background

2.1 The ISO 26262 standard

Recently, concerns about quality are driving the vehicle industry such as Scania to start looking seriously at approaches to improve and guarantee quality software development. The increasing use of electronic systems becomes one of the key reasons for defects in vehicles that affect their safety. Because of this, Scania and other vehicle companies are looking to standards such as ISO 26262 to help them meet the specific requirements of the electrical systems.

The ISO 26262 standard provides a risk-based approach for determining Automotive Safety Integrity Levels (ASILs). There are four levels (A-D) of ASILs to specify the necessary safety measures in order to avoid residual risks of one system component or the whole electronic system, where D is the highest level. The ASILs are assigned based on the risk of a hazardous event occurring based on the following attributes: (1) exposure (2) severity (3) controllability. ISO 26262 translates these software event levels into software specific objectives that must be satisfied during the software development process.

During the development process, requirements traceability is widely accepted to ensure that all requirements are implemented and that all the development activities can be traced back to one or more requirements. The ISO 26262 standard takes this one step further and requires bidirectional traceability having constant emphasis on the need for the derivation of one development tier from the one above it.

2.2 OSLC

Open Services for Lifecycle Collaboration (OSLC) is a standard for software integration processes. OSLC makes it easier for independent tools and software to communicate and share data with each other. This standard builds on the W3C Resource Description Framework (RDF), Linked Data and REST.

---

3. [http://open-services.net/resources/tutorials/oslc-primer/what-is-oslc/](http://open-services.net/resources/tutorials/oslc-primer/what-is-oslc/)
4. [https://www.w3.org/RDF/](https://www.w3.org/RDF/)
The integration becomes enabled by linking data from different resources. In the integration, resources are considered as RDF properties, the HTTP requests and response model is used for operations on resources, and linked data is the format of data [1].

In Scania, the documents for requirements, architectures, specifications, etc are separated. In order to fulfill ISO 26262, they require an OSLC platform where all the data and tools are linked together. This platform is named Espresso 2 (Figure 1.1).

2.3 Semantic Web

The number of web documents is increasing by orders of magnitude, while the efficiency to filter out desired information has been significantly improved. Although search technologies have been improved significantly, they are still not sufficient to help users deal with enormous data volumes efficiently. This is because the web documents are designed originally for human readers so they vary in both formats and structures. Therefore it is very hard for machines to process and understand web data.

In 1998 Time Berners-Lee, also the inventor of the world wide web, proposed the concept of the semantic web [2]. The core concept is to make computers understand the web documents by adding meta data, so that the world wide web will become a media for interchanging information.

2.3.1 The semantic web layer cake

The semantic web actually contains and uses multiple existing technologies. Back to the time when Berners-Lee proposed the semantic web concept, XML had already been widely used [2] and OWL⁵ (Web Ontology Language) and logic had been researched for decades. Berners-Lee creatively combined them, and designed the semantic web stack (Figure 2.1) having the following layers.

---

⁵https://www.w3.org/OWL/
1. The Unicode and URI layer: This is the bottom layer. Unicode is used to define international general character set. In the semantic web, everything that needs description is called a *resource*, and every single resource has a unique URI (Uniform Resource Identifiers).

2. XML +NS + XMLs layer: This is the fundamental descriptive language layer. XML is the basis for the semantic web layers, as it was one of the most widely used network tabbed description language. However, JSON is getting more and more popular, and one of the reason could be the widely usage of Javascript, which native recognizes JSON. In addition, JSON has a better performance than XML for data interchange [3]. Both XML and HTML are designed for marking up text and are not well suited for representing data, while JSON is designed as a very simple representation of data.

3. RDFS layer: RDFS (RDF Schema) has the same syntax as RDF. The RDFS layer offers the possibility to manually classify web resources, and it is fundamental in the semantic web. It is an extension of the basic RDF vocabulary [4].

4. Ontology vocabulary layer: This layer defines additional semantics as an advanced knowledge representation languages for defining ontologies. There may be different vocabularies which can be used within different applications in the context of the semantic web.

### 2.3.2 Semantic search engines

Search engines are computer programs that can search information by giving specified keywords and return a set of data where the keywords are found ⁶. In the semantic web, the search results should not only contain media objects (such as Web pages, images, etc) but also people, places, and events [5].

---

2.4 Linked data

In the semantic web everything is data. The most significant difference between the regular web and the semantic web, is that in the semantic web the data is organized for being machine-readable, while the regular web is organized for being displayed in web browsers. In order to convert everything on the web to machine readable form it is necessary to have a standard format. This standard format should be accessible and manageable by semantic web technologies, which are provided by the World Wide Web Consortium (W3C), such as RDF, OWL 7, SPARQL 8, etc. The term linked data is used for the standardized format for publishing and connecting structured data on the semantic web [6].

2.5 RDF

Resource Description Framework (RDF) is a W3C standard for data interchange on the web. RDF breaks down knowledge into sentences called triples, the atomic data entity having three parts: subject, predicate, and object. A triple is a statement linking an object (the subject) to another object (the object) or a literal, via a predicate. For instance, in RDF format, "Brussels is the capital of Belgium" could be represented as a statement where Brussels is the subject, capital is the predicate, and Belgium is the object. The subject could be a URI, which is used to identify a resource, a reference, or a blank node, which is an RDF node that does not contain data but could be parent of a group of data 9. The predicate is a URI reference and the object could be a URI reference or a literal. In the example, the URI, http://example.org/place/Brussels could identify the subject, http://example.org/properties/capital could identify the predicate, and the string capital is a literal node. The representation of the triple as a graph structure is shown in Figure 2.2.

2.5.1 Database for RDF

An RDF Database is needed for storage and retrieval of triples. Triple stores are DBMSs which are used for storing RDF databases of triples. On the other hand, data can also be stored in a relational DBMS, while having a mapping layer above it to map relational tables to RDF triples. In this project, a triple store is the choice.

A triple store is one type of graph databases which has grown rapidly lately[7]. They have the ability to ingest semi-structured data and provide flexibility

7https://www.w3.org/TR/owl-ref/
8https://www.w3.org/TR/rdf-sparql-query/
Figure 2.2. A Triple sample: Brussels is the subject node which is connected to Belgium as object node by the predicate arrow which is is the capital of.

with respect to schema changes. One of the most important advantage of triple stores is the cost for data integration, management and query definition is much lower than other kinds of databases 10.

2.6 Solr

Solr (pronounced "solar") is an open source, RESTful search platform for full-text indexing running on the Jetty server 11, which is an open source Java project providing HTTP servers and clients. Solr provides a significant number of features to facilitate and increase the speed of finding information. It provides advanced full-text search and matching capabilities, extremely large scalability, re-balancing, fault tolerance, near real-time indexing capabilities, database integration, and faceted search that categorizes search results into different groups to narrow them down. Solr can be used by any kind of web-applications that need search functionality. Solr is Java based and has powerful XML-based configuration that eases customization by web developer. 12

2.6.1 Solr Architecture

Solr is based-on another open source search server called Apache Lucene 13, a Java library with indexing and search technology capabilities. Both Solr and

---

12http://lucene.apache.org/solr/features.html
13https://lucene.apache.org/
Lucene are managed by Apache Software Foundation\textsuperscript{14}. Figure 2.3 shows the architecture of Solr\textsuperscript{15}.

### 2.6.2 Searching in Solr

There are three components in Solr to control the searching process. The following briefly describes each of them.

1. **Request Handler**: The request handler is the most important component of Solr. Solr includes many different types of request handlers. Some of them are responsible for processing search queries and some are responsible to manage tasks such as index replication.

2. **Query Parser**: The query parser is a request handler used for processing a search query. The standard query parser, DisMax, and eDisMax are query parsers used by Solr. As default Solr is using the standard query parser, which focuses on precision of results. The DisMax query parser and eDisMax both focus on being error tolerant, which means that their

\textsuperscript{14}https://cwiki.apache.org/confluence/display/solr/Getting+Started

\textsuperscript{15}http://riiteshbhhachawat.com/liferay/apache-solr-liferay/
Figure 2.4. An overview of steps involved in a Solr search

main focus is on avoiding to show syntax errors to user.

3. **Response Writer**: The response writer determines the format of search results. The two most popular response writers are the `XMLResponseWriter` and the `JSONResponseWriter`, returning the search result in XML format and JSON formats, respectively. Figure 2.4 illustrates an overview of the steps and components involved in a Solr search.\(^{16}\)

Figure 2.4 shows an overview of steps involved in a Solr search. A Request Handler is responsible to get users request and process it. Query Parser parses the query for syntactical errors and then translate it to a format which is understandable by Apache Lucene. Result of the search query then will send back to user by Response Writers. There are different Response Writers to handle different formats of result such as XML and JSON.

### 2.6.3 Three configuration files in Solr

The whole index process on the Solr side is mainly controlled by these three files. They are:

\(^{16}\)https://cwiki.apache.org/confluence/display/solr/Overview+of+Searching+in+Solr
1. **Solr.xml**: This is the first file that Solr looks at. It contains general configuration information such as locations of Solr’s cores. A core in Solr defines an index of the text inside a document. The rest of the configuration should be done in each core.

2. **Solrconfig.xml**: Most of the configuration sections such as data directory location, cache parameters, request handlers, and search components of Solr should be done in this file.

3. **Schema.xml**: This is an xml file used by Solr to declare which fields should be indexed, the type of the indexed data, the importance of the each field, and so on. If Solr is running in distributed mode, each Solr core could have a separate schema.

By creating a new Solr instance, if the user does not manually define a schema, the default schema (managed-schema) will be used for indexing. Referring Solr to use a manually defined schema can be done by renaming the managed-schema to schema.xml. After that, in the file solrconfig.xml one should replace ManagedIndexSchemaFactory with ClassicIndexSchemaFactory to tell Solr to use the manual schema.

The schema and solrconfig.xml are highly depended on each other. All of the fields that are used in different components of solrconfig.xml, as explained above, should be defined in the schema as well. The schema contains definitions of fields, data types, and analyzers.

The field types in a schema, describes how Solr should use the data and how to query it. The types supported by Solr are int, float, long, double, date, string, text, and location. In a Solr schema there are three different fields: Fields, Dynamic Fields and Copy Fields.

- **Dynamic Fields**: In some cases a client cannot be sure that all the fields are indexed. In some other cases there is a common pattern for the names of most of the fields, for instance there are huge amounts of words ending with suffix "y". In these situations, Solr recommends to use dynamic fields for indexing them. The dynamic field could be treated as a regular expression filter where any incoming field that matches the expression will be indexed. For instance if a dynamic field is named "/*_y", Solr

---

17[https://cwiki.apache.org/confluence/display/solr/Nodes%2C+Cores%2C+Clusters+and+Leaders](https://cwiki.apache.org/confluence/display/solr/Nodes%2C+Cores%2C+Clusters+and+Leaders)
will index all the field names having character "y" at the end.

- **Copy Fields**: A copy field is used when a single field can contain different data types. In other words, it will put one or more different source fields in one single field. An example is when we are going to use multiple fields for spell checking. Instead of defining all the fields one by one in solrconfig.xml, we declare one copy field which comprises of all of them and use this copy field in solrconfig.xml for spell checking purpose.

- **Analyzer, Tokenizer and Filter**: An Analyzer is a feature provided by Solr that breaks down a text file into tokens. It consists of two components called tokenizer and filter, where the tokenizer breaks indexed fields into lexical units, or tokens, and the filter can keep, modify or delete the received tokens by a defined pattern. A chain of processing between these two components makes the analyzer work (see Figure 2.5). They define how Solr should index textual data.

### 2.6.4 Indexing RDF data with Solr

In database systems, indexes are used in order to improve the performance of querying information\(^\text{21}\). An index (IDX) is a data structure to speed up data access. In this project Solr is chosen in order to implement a search and browse platform.

There are some search engines that are specifically designed for the semantic web [8, 9], but none of them are supported as well as Apache Solr. The main reason why Solr is chosen as the search engine in this project is because it is widely used by big enterprises and is a mature and reliable product. It is well supported by a developed community, so it is easy to get solutions of common issues. Solr is also very well documented where most problems have detailed descriptions. However, it does not directly support indexing and searching RDF data, so different approaches and different combinations of tools were tested in order to utilize Solr as an RDF search engine. In order to index RDF, one of the following alternatives were available:

- **Convert RDF data to JSON-LD**: One way to index RDF data with Solr is to change the format of all data to JSON-LD (JavaScript Object Notation for Linked Data)\(^\text{22}\), which is an extension to JSON (Java Script Object Notation) for the purpose of easy access using a human readable data representation. JSON-LD represents linked data in JSON, which is supported by search engines such

\(^{21}\)https://www.techopedia.com/definition/1210/index-idx-database-systems

\(^{22}\)http://json-ld.org/
Figure 2.5. Analyzer, Tokenizer and Filter: Tokenizer produce a sequence of Token objects (a TokenStream) from input text. Filter looks at each token in the stream sequentially and decides whether to pass it along, replace it or discard it. A sequence of tokenizers and filters forms an Analyzer.
as Solr. This approach was not chosen because, the process becomes complicated. The conversion from relational data to RDF is required followed by the conversion from RDF to JSON-LD, which needs extra disk space to store the same data in another format.

- Data import handler in Solr:
  Solr can easily work with relational databases, such as MySQL or Oracle. This is because Solr provides a configuration file (solrconfig.xml) for users to use and enable Solr to index the relational database automatically. The user can define a request handler to process queries to retrieve the data to be indexed. In addition, the user can also define which field (table column) should be indexed. However, Solr with back-end relational databases does not work straightforward for triple stores (RDF databases) because SQL queries cannot be used to retrieve data from triples stored in a relational RDF database. Apache Solr is not a search engine that is specific for the semantic web so it does not support searching triple stores by itself. However, some organizations try to make Solr suitable for indexing and searching RDF data format for instance Apache Jena and SolrRDF. In this project we used the Apache Jena approach in order to use Solr as the search engine.

2.7 Apache Jena

Jena is an open source framework for implementing applications based on the semantic web and linked data. It consists of several APIs interacting together to process RDF data. Some of APIs provided by Jena are: RDF API, the Sparql endpoint Fuseki, Triple store (TDB), and text search. Following is a brief description of Fuseki and TDB which are APIs used in this project.

Fuseki is a RESTful SPARQL server and has a user interface for server monitoring and administration. It can run in three different ways: stand alone server, Java web application (WAR file), or as an operating system service.

The triple store component in Jena is called TDB. Accessing and managing RDF data in TDB can be done by either command line scripts or the Jena API.

The easiest way to index RDF data with Solr is by using Jena Assembler Description. The assembler is a Turtle file, which is a textual representation of RDF data.
Figure 2.6. Jena Assembler is a textual representation of an RDF graph. It describes a text dataset including the RDF storage and text index. The technology used for indexing (Lucene or Solr) with the details and the fields chosen for indexing should be used by the text index.

In a text index description the search system, here Solr, is defined. Therefore the index description should link to the Solr core which is used for indexing. In the *entity map* the fields that need to be indexed are defined. These fields should map to the equivalent fields in the Solr schema file.

### 2.8 UI design for searching and browsing

User interface (UI) design is a practice whose technologies are encompassed by the field of Human-Computer Interaction (HCI). The HCI field studies how people are interacting with user interfaces. The design of a UI is an iterative process. The UI designer’s work iteration is to study how users use the UI, including how do they think about, respond to and use technology, and then redesign the UI incrementally [10].

---

27https://www.w3.org/TR/turtle/
The design process can begin by determining what exactly the user wants to do, and then devising the interfaces that can help users achieve their goals. Depending on the kinds of data that the user wants to retrieve, the process of information retrieval (IR) can be quite different [11]. Here IR means the activity of obtaining information resources relevant to information in a collection of information sources. For instance, if a user wants to search for latitude and longitude of a specific place then the search result could be unique, but if the user searches for requirements of a specific function, the search result could be a set of files. Thus, in the UI design process, the first priority task is to create a proper interface type for the specific usage scenario.

Eyal Oren and his colleagues [12] have identified four existing interface types for navigating RDF data: (1) keyword search, (2) explicit queries, (3) graph visualization, and (4) faceted browsing. In this project, one goal is to design a UI for search and browse RDF data which should be (1) scalable, (2) generic, (3) not depending on a fixed schema, and (4) allow exploration of the dataset without knowledge of its structure. There is no single interface type that fulfills all these requirements: the keyword approach is sufficient for simple information lookup, but not for investigating and learning [13], writing explicit queries is difficult for mechanical engineers in Scania, and graph visualization is not scalable [14].

A combination of two interface types are used for this project. The keyword approach is used as the first HTML page of LD Search & Browse App and faceted browsing is used as the second HTML page. With this combination, the users at Scania can use keyword to narrow down search results to the domain of their interested information and use the facets to precisely get the data.
3. The Search and Browse System

LD Search & Browse App is a RESTful web application for retrieving information in RDF format. The inspiration is from Google for searching and Wikipedia for browsing and rendering information in a user friendly way.

3.1 User Interface

When users enter the address of the LD search & Browse application in a browser, the first screen will be shown (Figure 3.1). This screen is designed as simple as Google. It contains a text-box for keyword input and a button to fire search. As soon as the user hits the search button, the search process will be started. After the query has been processed and information is found, the second screen will be presented (Figure 3.2). This screen contains a number of relevant results. It is possible for the user to filter the search result further by using the filter in both the left and top facet boxes. As it is shown in Figure 3.2 in the left side of the screen, there are different options that help the user to narrow down the search results. For instance, the user can choose one of the addresses to limit the search results to those data that exactly have the same address. When the user chooses one of the results and hits the link, s/he will be navigated to the third screen that contains detail information about specific data (Figure 3.3).

3.2 System Implementation

The developing environment was Windows 7 Enterprise edition from Microsoft 1 with service pack 1. Eclipse for Java EE developers 2 64 bit edition was used as integrated development environment (IDE). Apache Solr version 5.5.0, Apache Fuseki version 2.3.1, and Apache Jena version 3.1.0 were used for this project.

3.2.1 System Architecture

The application is based on the MVC (Model View Controller) model. In Figure 3.4 the architecture of LD search and browse application is shown.

1https://www.microsoft.com/
2https://www.eclipse.org/downloads/
Figure 3.1. First screen of LD Search & Browse App

Figure 3.2. Second screen of LD Search & Browse App
Figure 3.3. Third screen of LD Search & Browse App

The top layer is the view layer, which is the user interface of the application. Any user action in the user interface such as hit a button or click on a link is considered as an HTTP request. The middle layer is called the controller, and is a Tomcat web application server that gets this HTTP request from the view layer. Then Tomcat sends the user request to the model layer in order to handle it. In the model layer the data is stored in the TDB triple store. Since the index file of the original data is kept by a Solr server, we put Solr in the model layer as well. However, the Solr server Jetty, can be considered as a controller as well. After Solr handles a user request, the controller fetches the data and sends it back in the form of a HTTP response to the view. This response renders as the search result to the endpoint user. Since the controller uses http request and http response for communication between the layers, the architecture style of the application is RESTful.

3.2.2 The user interface

Nowadays almost everyone uses Google to find any kind of information. It is simple and powerful. Due to this fact we decided to have a user interface similar to Google so everyone know how to use it without any need to put illustration or guidance for our application.

3http://rest.elkstein.org/
Figure 3.4. System Architecture of LD Search & Browse App
A search engine should include elements to:

1. Enter the search keyword: The first page of UI implemented in this project is simple. It includes the logo, search bar, and search button (Figure 3.1). This page includes two important features that made the application more tractable for the user:

   - **Keyword Suggestion:** Having this, when user start typing a search keyword, a popup will be displayed below the text as the user types. This feature analyzes text while the user is typing and compares it to existing indexed data. This means that the user can quickly select existing content from the popup so that s/he does not have to finish typing it.

   - **Spellcheck:** This feature locates misspelled words typed by user and notifies the user of the misspellings. Depends on spellchecker, it could either correct the word or ask a clarification question like "Did you mean ...". In this project the second is used.

   In order to provide these two features, a search component for each should be added to solrconfig.xml. Each of them takes several configuration parameters. The suggest component has several parameters. The field parameter defines the field(s) from an index as a suggestion field. In this application, the field is used as a suggest field called suggestion, which actually is a compound field (see 2.6.3) in schema.xml comprised of several other fields. The lookupImpl parameter declares how terms are found in a suggestion dictionary. The parameter dictionaryImpl declares how terms are stored in the suggestion dictionary. Then a request handler must be added to solrconfig.xml to allow the configuration of default parameters to serve as suggestion requests. The request handler definition must incorporate the "suggest" search component defined previously. Some parameters that this request handler needs are suggest which should always be set to true and suggest.count which specifies how many suggestion should be returned by Solr (Figure 3.5).

2. Display the list of results: The second screen after clicking on the search button in the first screen, should show an overview (snippet) of most relevant results. The snippets should be enough rich in content so the user can select an item with confidence. To reach this goal, snippets should show enough information to give a strong scent of information

---

4https://cwiki.apache.org/confluence/display/solr/Suggester
and also should be easily scan-able. There are some general features that a search engine UI should have to be attractive and user-friendly. Some of the features that have been used in this project are listed:

- **Highlighting the keyword in the snippet**: Bolding the search key in the snippet of a search result improves the search experience and it is less time consuming compared to showing the search result only so that user could know why the document match the search keyword. Solr supports this feature by having a request handler that includes different parameters such as `hl` which enables request handler to generate snippet in the response, `hl.fl` where the value(s) of this parameter are the field(s) that should be shown in snippets. Using "*" to define all fields as snippet is acceptable. In this case the value of `hl.requireFieldMatch` parameter should be set to true.

- **Static Filters**: Filters that can be used by users to narrow down search results. In this project a set of filters provided by Scania is used. These filters are common in all tools and frequently used by engineers. Independent of the search keyword, users can always

---

5http://www.wqusability.com/handouts/search-ui.pdf
6https://codeforgeek.com/2014/12/highlight-search-result-angular-filter/
see these filters in the search result screen.

- **Faceted Navigation:** To enable users to browse large set of information quickly and intuitively, faceted navigation is used. Faceted navigation or facet search means to categorize a huge amount of data to different groups in order to drill down the search result. Semantic web represented in RDF is very voluminous and highly interconnected so having scalable techniques to navigate graph-based data is necessary [12].

This project aims to gather data from different tools and databases where each one of them has its specific users with different purpose. It is important to organize the search result in meaningful groups. One way to do this is **faceted classification**. A faceted classification is a set of grouped data which have been organized into different categories [15].

**LD search & Browse App** includes both static and dynamic faceted filter functions. Static facets are those data classes that are predefined and frequently used by engineers, for instance *SOP* is a date from when a specific function of a vehicle becomes valid, while *date*, and *product* are using very often by engineers.

Dynamic facet is a dynamic list of classes that are used to classify the searching data. It is called dynamic because the facets’ list will change dynamically based on what data the users are searching for (Figure 3.2).

3. Display detail information about search keyword: The next screen after user clicking on one of the search results contains complete information about the requested search keyword. One of the most interesting features of this screen is that when any part of the text has additional information available in the database the text will be link-able, and the link will navigate user to the detail information screen about this specific data.

3.2.3 Middle component

The middle component or middle layer has five functions: (1) receive user’s input from UI, (2) convert the input into a query, (3) send the query to Solr, (4) receive the search results returned by Solr, and (5) reformat the search results and send them to the UI. This component is mainly implemented by using

7http://alistapart.com/article/design-patterns-faceted-navigation
Apache’s Jena library\(^8\).

The *query composer* is one of the two subcomponents of the middle layer. It is responsible for the above functions (1), (2), and (3). It creates a client of Solr which receives the keyword input from the UI through a *doPost* method, and use Edismax to parse/compose the keyword\(^9\). After the query is ready, it will be sent to the Solr server.

The other component is called *SolrJ*, which receives search results from the Solr server and prepares them in a specific format. In this project the search results are organized in the order of data information, facets information, and highlight information.

### 3.2.4 Backend

On the backend side, there are the Solr search engine and Triple Database (TDB) within Apache Fuseki running on Jetty. Jetty comes with Apache Solr and Fuseki and it needs no extra configuration to run them.

**Solr configuration**

Solr is a very powerful search engine [16], it is quite scalable by using the cloud model and running on multiple servers. In this project only the single core model is used for the prototype. A Solr server can be started by using command:

\[
\text{solr5.5.0}\bin\text{solr start } -p 8983
\]  \(3.1\)

When the Solr server is running, a new core named *test1* can be created by using command:

\[
\text{solr5.5.0}\bin\text{solr create } -c test1
\]  \(3.2\)

After the core is created, the configuration files will be automatically generated and stored inside the core folder. *schema.xml* and *solrconfig.xml* are the most important configuration files. The *schema.xml* file contains all of the details about which fields should the core contain, and how those fields should

\(^8\)https://jena.apache.org/
\(^9\)https://cwiki.apache.org/confluence/display/solr/The+Extended+DisMax+Query+Parser
be dealt with when they are added to the index or when querying the fields. In this project, a series of fields were defined. Some fields need to be splitted, tokenized, and matching of words that are defined as text. Some fields such as ID or pure numeric data are defined as strings, because the string type of field will not be analyzed, so they can only be searched by a totally matching.

Another important configuration file is solrconfig.xml. It is the file that has the most parameters affecting Solr itself. As what has been stated above, it can define which kind of schema does Solr use, where to save index files, or which fields should be used as filters. In this project, Address and Id are chosen as facets to narrow down the search results.

Fuseki configuration

Apache Fuseki in this project runs as a standalone server. Data administrators can use the Fuseki UI to load the data file, and the added data will be automatically indexed into a predefined Solr core.

This automation is implemented by using the configuration file of Fuseki called jena-text-config.ttl. In the entMap section of jena-text-config.ttl, the properties of the RDF data that need to be indexed should be defined and assigned a tokenizer. Every field that is defined here should also be defined in the Solr schema.xml. A Solr core’s address was provided to tell Fuseki where to index the data.
4. System Evaluation

4.1 Usability Evaluation

In the usability test, a group of six users who are working in Scania as engineers were randomly chosen to use our prototype. There is no specific task of this interview, the goal is to check if users can search and get the information related to their work and make a comparison with an existing tool in Scania. After the interview, a questionnaire was provided for them to fill in.

The test interview started with an example of search keywords, the purpose is to show the format of search phrases. Then they used the prototype as a replacement of an existing search tool. During the search, user told what they are thinking about the prototype. When the search was done, user was asked to fill in a questionnaire.

Information Search Completion Result

According to the observations, five out of six participants successfully searched and retrieved what they were searching for. Some of the users enjoyed using this specific format of search, because they could get the results much quicker than with the existing tools. For example, one of the user wanted to use a combination of several conditions to search, which is not supported very well in the current tool, but with this prototype, the user could get a precise and quick response. The only user who felt a little frustration was because the prototype can not provide a free text search over subject names. This is because of two reasons: 1: The subject URI is not provided by the database engineers and there is no schema available. 2: The configuration of the Solr is still in progress.

System Usability Scale Questionnaire Result

At the end of the test, each user was asked to fill in the System Usability Scale (SUS) Questionnaire [17]. The standard SUS consists of the following ten questions (odd-numbered questions worded positively; even-numbered questions worded negatively) [18] (see also appendix A)

1. I think that I would like to use this system frequently.
2. I found the system unnecessary complex.
3. I thought the system was easy to use.
4. I think that I would need the support of a technical person to be able to use this system.
5. I found the various functions in this system well integrated.
6. I thought there was too much inconsistency in this system.
7. I would imagine that most people would learn to use this system very quickly.
8. I found the system very cumbersome to use.
9. I felt very confident using the system.
10. I needed to learn a lot of things before I could get going with this system.

According to the study of SUS, a typical minimum reliability goal for questionnaires used in research and evaluation is 70 points.

Table 4.1. SUS Questionnaire Results. Maximum score of this method is 100.

<table>
<thead>
<tr>
<th>User</th>
<th>Q1</th>
<th>Q2</th>
<th>Q3</th>
<th>Q4</th>
<th>Q5</th>
<th>Q6</th>
<th>Q7</th>
<th>Q8</th>
<th>Q9</th>
<th>Q10</th>
<th>Score(x2.5)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td></td>
<td>67.5</td>
</tr>
<tr>
<td>B</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td></td>
<td>70</td>
</tr>
<tr>
<td>C</td>
<td>3</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td></td>
<td>70</td>
</tr>
<tr>
<td>D</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>3</td>
<td>1</td>
<td></td>
<td>72.5</td>
</tr>
<tr>
<td>E</td>
<td>4</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td></td>
<td>77.5</td>
</tr>
<tr>
<td>F</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>2</td>
<td>3</td>
<td></td>
<td>75</td>
</tr>
</tbody>
</table>

The average SUS score of this prototype is 72, which is above the minimum reliable score. In this case, the prototype has a good usability.

4.2 Comments and Suggestions from Scania Engineers

Here are some comments from engineers about the LD Search & Browse App:

User A

"I like the system, It is easy to use and much faster than the old system which is used to retrieve information."

User B

"I like the vision of application, however it still needs many improvements both in UI and functionality."

User C
"I would prefer that the search keyword is not case sensitive. It should be like google and no matter you write ID, iD, Id or id it returns the relevant results."

User D

"I really like the logic search of application. However it is not that much user interactive when we put extra space or when I miss to put one space it will not show the result."

User E

"I really like the application. It helps us to access to the data from all databases fast and easily. The UI still needs to be improve specially the faceted navigation. The logical functionality of search is really interesting. The is good if it could recognize between for example AllocationElement and AE. I mean it even user write AE an abbreviation of AllocationEllement the system show proper result to user."

User F

"I feel more confidence with the old tool. Because I just copy the information I need from my excel-sheet and paste it in the tool and I know where is my information because I use it almost everyday but this application return many other unnecessary results as well. By the way this application is user-friendly and easy to use in my opinion."
5. Conclusion and future work

In general, the results of the usability test are positive. Five out of six participants successfully retrieved their desired search results. In addition, their feedbacks during and after the tests were mostly positive. However, there are some problems exposed as well during the tests as discussed below.

According to the observations, only two users accepted the search syntax immediately. This is because these users are familiar with regular expressions. With the help of the search syntax they can do a complex queries such as:

\[
\text{Type:} message \text{ AND Name:} CO1 \text{ AND Freebits:} 4 \text{ AND CircleTime:} [10,100],
\]

which means retrieving a message from component CO1 with four free bits and a circle time is between 10 and 100 ms. Some users who are not so used to query syntax can only use a simple syntax such as:

\[
\text{Type:} \text{UseCaseItem}
\]

which means show all the data that is a type of UseCaseItem and will then browse the entire query result.

Some users do not like the chosen style of queries; they prefer using totally free text search. The reason is that using query syntax needs users who are familiar with both keyword expressions and the semantic of the data source. Another problem is that the syntax currently is case and space sensitive, which means that the users need to be very careful when they input the keywords.

In the implementation, the Solr default query syntax \(^1\) is very useful to query the current data with literal value properties since all the literal values are indexed. For example, if the user searches for a data which has a property called address, and its value range is from 0x01 to 0xff, the user can use the syntax and easily find the expected data. However, when the users want to search with abbreviations of property names or the subject names, the default syntax will not be helpful because the abbreviation names must be predefined inside Solr. After the project, abbreviation names will be defined by the database engineers and mechanical engineers, and when abbreviation list is prepared later by the engineers, the configuration of Solr could be improved so that users can search by using abbreviation names.

There are also some positive facts about the UI. Some users think the first screen is simple and easy to use because it is similar to the Google style.

\(^1\)https://wiki.apache.org/solr/SolrQuerySyntax
are able to start inputting searching keywords without any help. For the second screen, the users think the way that the snippets show data is quite understandable and brief. It is also very good that they can combine different facets to filter search results. On the third screen, there are links to documents that exist in some other databases. Thus, without this prototype the Scania engineers have to open several tools to search and browse data, but with it they are able to access all Scania data at once, and all the data are linked. The users were quite impressed about this feature.

5.1 Future work

A search and browse application for all the Scania’s data: Engineers in Scania are still working on converting data from all tools to RDF or Turtle formats due the goal of OSLC. In this project we have used data from two databases: WP6 and SesammTool. Since RDF data from different tools have different predicates\(^2\), Solr’s schema file should be updated to add new properties to be indexed.

Automodification for Solr’s schema and configuration file: As discussed above, by adding new databases, new properties will appear and modifications in the configuration files are necessary. In the the future, when the schema (RDFs, OWL, Schacle, ...) of data is prepared, it could be used to automate the process of adding new predicates to Solr’s configuration and schema files.

Improvement in UI: The user interface should be improved to become more adaptable with Scania standards. There were some suggestions for how to improve the prototype. For the search part, the syntax is not suitable for all the engineers, it is space sensitive, which means it has a high possibility that the search result is not what users are expecting. Another problem is that the user needs to have knowledge about the data before they can use the syntax. They have to remember the property names such as Fullname, CircleTime or Type, it is also important to write the case correctly, which is very inconvenient. In the future, it could be better that the prototype can provide precise search results with free text search.

Dynamic Faceted search based on data-mining algorithms: The application should be smart to show facets based on search keyword to users and avoid to show unnecessary facets. This is doable by using the ClusteringComponent from Solr.\(^3\)

\(^3\)https://cwiki.apache.org/confluence/display/solr/Result+Clustering
Authentication and Access Control: The data from different databases has different authorities. It is very crucial for Scania that every engineer only is able to access the information which they are granted to. The access control concept for RDF data is a broad concept and needs investigation.

Performance: Since the project described in this report is a prototype, the performance for multiple users and web response time has not been considered in the design. The ability to scale up the system for serving all engineers in Scania should be investigated.
References


Appendix.
Appendix A: SUS-Questionnaire
System Usability Scale


1. I think that I would like to use this system frequently
   - Strongly disagree
   - Strongly agree

2. I found the system unnecessarily complex
   - Strongly disagree
   - Strongly agree

3. I thought the system was easy to use
   - Strongly disagree
   - Strongly agree

4. I think that I would need the support of a technical person to be able to use this system
   - Strongly disagree
   - Strongly agree

5. I found the various functions in this system were well integrated
   - Strongly disagree
   - Strongly agree

6. I thought there was too much inconsistency in this system
   - Strongly disagree
   - Strongly agree

7. I would imagine that most people would learn to use this system very quickly
   - Strongly disagree
   - Strongly agree

8. I found the system very cumbersome to use
   - Strongly disagree
   - Strongly agree

9. I felt very confident using the system
   - Strongly disagree
   - Strongly agree

10. I needed to learn a lot of things before I could get going with this system
    - Strongly disagree
    - Strongly agree