THE UNIVERSITY OF AUCKLAND
NEW ZEALAND

School of Computer Science
COMPSCI 791

SPARQLing Access to Māori Perspectives of Library Data

Student: Channing Wang
Supervisor: Sebastian Link
Date: July 17, 2022

A dissertation submitted in partial fulfillment of the requirements for the degree of Master of Professional Studies in Data Science, The University of Auckland, 2022.

Abstract

As Matariki has been celebrated across New Zealand as a national holiday, the revitalisation of Māori culture sees a unstoppable trend. This report will guide you through building a modern data pipeline from scratch to integrate thesaurus and bibliographic data for their indigenous heritage, which will help towards restoring their culture. We will use different technologies to make it happen, ranging from the latest Semantic Web tool set to traditional web development techniques. In the end, a web app is created, paving the way to take this ongoing project to the next level.

Acknowledgement

I would like to thank my supervisor Professor Sebastian Link for taking me on the project. During the project, he not only gave me enough freedom to build things up in my own way and in my own time, but also generously offered insightful advice whenever I needed it. Not to mention my friends and family, having them by my side was an encouragement to push through difficult times.

Contents

1 Introduction

The project *Ngā Ūpoko Tukutuku / Māori Subject Headings (MSH)* jointly sponsored by LIANZA, Te Rōpū Whakahau, and the National Library was released in 2006 to provide a structured pathway to subjects that Māori customers can relate to and use to find materials in library. [1]

The most significant proceeding from MSH is the thesaurus developed from the Māori perspectives of library data, and up to this day, new terms such as COVID-19 / KOWHEORI-19 are still being actively added to the collection. On the other hand, librarians contribute to that knowledge base by tagging relevant published materials in / about Te Reo Māori with the terms from the thesaurus. Over time, two silos of data become more separated - while the thesaurus data is maintained by the working group from MSH, the bibliographic data is sourced from the records that librarians work on from different parts of New Zealand.

Our project *SPARQLing Access to Māori Perspectives of Library Data* addressed the problem by integrating two silos of data into a graph database system. In the following sections, we follow the modern data stack to represent data, model data, and eventually access data. We will go into details of what is happening at different stages of the project, from theoretical bases to practical problems.

In Section 2, we will talk about how we translate the thesaurus data and bibliographic data into Resource Description Framework (RDF) documents that can be later imported into our graph database AllegroGraph. Compared to traditional relational database approaches, which put a strain on performance when it comes to queries that involve heavy joins, a graph database offers an alternative where a data point is naturally connected to other data points, making it easier to execute a query within a linked structure. That is the case for our data, and we assume better performance than that in a relational database.

In Section 3, we model data using Web Ontology Language 2 (OWL) and take advantage of the semantic language to make inferences on the data. The idea behind that is RDF Schema (RDFS), which is mostly dealing with a simplified mechanism to describe the hierarchy for classes and properties, is not sophisticated enough to describe our data at hand. OWL extends the vocabulary in RDFS to allow for the expression of complex relationships and more precise constraints, which is more valuable today when proper data modelling is often neglected at the very beginning, leading to data debts for the future data engineers. OWL makes it possible even to extend the model with rich features it offers.

In Section 4, we query our graph database via SPARQL to provide basic functionality. Apart from that, a web app is also developed to better serve the needs of our intended users who are non-IT professionals like librarians or the working group for MSH. We use SPARQL and ASP.NET Razor Pages for SPARQL is recommended by the World Wide Web Consortium (W3C) as a query language for RDF and ASP.NET Razor Pages is supported by Microsoft, we can benefit from a mature community for the sustainability of our project.

In Section 5, we highlight what we have achieved in the project and limitations that can leave for the future work.

In Section 6, we discuss about the issues with the projects and the overarching demands that we can see from our users.

Please feel free to navigate to the topic of your interest for a further read.

2 Representing Data

2.1 XML

XML is a document format to facilitate the exchange of heterogeneous data. It consists of the basic components as below. [2]

- Element, e.g. `<elt_name>content</ele_name>` where the content could be one of
  - textual content
  - nested element
- Attribute, which can be attached to an element as a name-value pair,

  e.g. `<elt attr_name="value">...</elt>`

2.1.1 XPath

XPath describes paths in an XML document, and the retrieval of nodes that match those paths. It is made up of consecutive steps in the form of [/]step1/step2/. . . /stepn where each step is to [3],

1. start with the context nodes resulted from the previous step,
2. go in the direction of the axis,
3. and use node-test and predicates as additional conditions to select nodes.

2.1.2 XQuery

XQuery takes XPath one step further mainly by leveraging the power of the FLOWR expression as below [4].

- *For* to iterates over sequences
- *Let* to define and bind variables
- *Where* to apply predicates
- *Order* to sort the result
- *Return* to construct a result

2.2 RDF

2.2.1 RDF Graph

The fundamental idea of RDF is the triple of subject, predicate, and object. [5] In RDF, those are referred to as the resources denoted by Internationalised Resource Identifiers (IRIs). We can think of an IRI as an extension to web address to allow more characters.

As shown in Figure 2.1, subjects are represented as node elements (the nodes in green at the tails of arrows), objects are represented as node elements (the nodes in green at the heads of arrows) or literals (the rectangles in yellow at the heads of arrows), and predicates are represented (arrows) as property elements. We can easily pick up triples like the node element with the IRI *http://www.w3.org/TR/rdf-syntax-grammar as a subject, the property* element labelled with the IRI *http://example.org/terms/editor* as a predicate, and the node with no IRI as an object, or likewise, the node with the IRI *http://www.w3.org/TR/rdf-syntax-grammar* as a subject, the property element labelled with the IRI *http://purl.org/dc/elements/1.1/title* as a predicate and the literal node with the textual content *RDF 1.1 XML Syntax* as an object. [6]

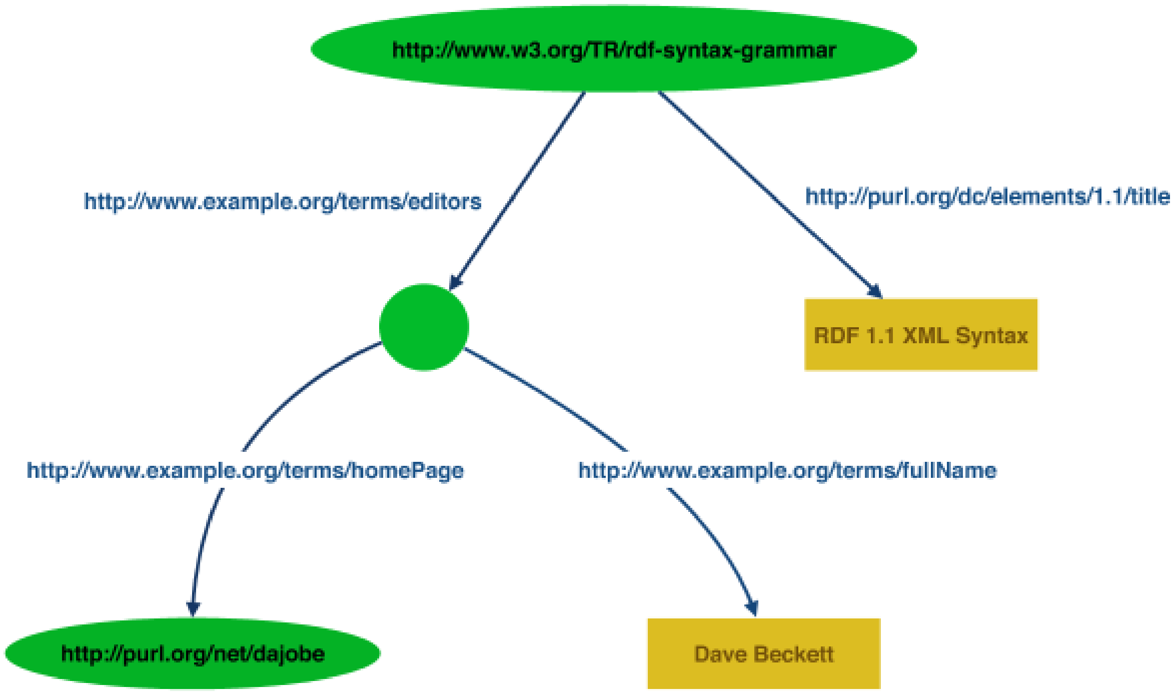

http://www.w3.org/TR/rdf-syntax-grammar
http://www.example.org/terms/editors
http://purl.org/dc/elements/1.1/title
RDF 1.1 XML Syntax
http://www.example.org/terms/homePage
http://www.example.org/terms/fullName
http://purl.org/net/dajobe
Dave Beckett

Figure 2.1: RDF Graph

2.2.2 RDF/XML Document

Intuitively, we can easily construct an RDF graph to represent the data, but it must be translated into an RDF document so that machines can understand it. An RDF document can come in many formats, i.e., turtle document, OWL/XML document, function syntax documents, Manchester syntax document, or RDF/XML document. Our approach follows closely with the syntax of RDF/XML document as RDF/XML document is the earliest standard of all and is supported by all RDF-based databases, plus that fact that it is also in XML gives us more edges to make use of XML technologies.

Conventionally, there is a way to abbreviate the namespace part of an IRI. The namespace ends with path as shown in Figure 2.2, and then we give it a namespace prefix, follow by a colon and the remainder of IRI taking away namespace (local name). Say from the RDF graph in Figure 2.1, we extract the namespace *http://example.org/terms/*

from the IRI *http://example.org/terms/editor*, give a namespace prefix *ex*, followed by *:* and the localname *editor*, we can get *ex:editor* as the prefixed name for the IRI *http://example.org/terms/editor*.

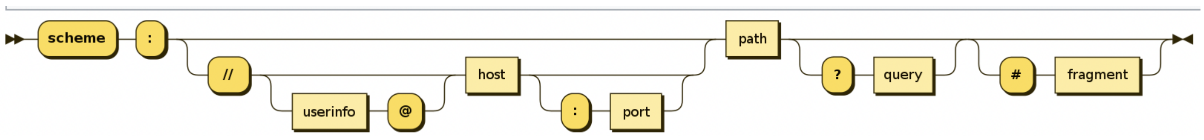


Figure 2.2: IRI

The other two tricks are for empty property elements and typed node elements [6].

- Empty property element means the node element with no additional property element. We can use the IRI of the empty node element as the value of an attribute *rdf:resource*, so that the form can be expressed more concisely.
- For a typed node element, which is referred to as the node having *rdf:type* predicates from subject nodes. We can shorten it by replacing the node element name with an IRI of the type relation. Since it is not a value of an attribute, it can be shortened one more time by giving it a prefixed name.

Now that we know about the syntax of an RDF/XML document, we only need to nest the resources in the same order they come in triple to write up the document. For example, the RDF graph mentioned above can be translated into the RDF/XML document shown in Figure 2.3

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
            xmlns:dc="http://purl.org/dc/elements/1.1/"
            xmlns:ex="http://example.org/stuff/1.0/">

<rdf:Description rdf:about="http://www.w3.org/TR/rdf-syntax-grammar">
  <ex:editor>
    <rdf:Description>
      <ex:homePage>
        <rdf:Description rdf:about="http://purl.org/net/dajobe/">
        </rdf:Description>
      </ex:homePage>
      <ex:fullName>Dave Beckett</ex:fullName>
    </rdf:Description>
  </ex:editor>
  <dc:title>RDF 1.1 XML Syntax</dc:title>
</rdf:Description>
```

Figure 2.3: RDF Document

2.3 Thesaurus Data

First, we need to transform the thesaurus data. We can get a hint of its structure from how-nga-upoko-tukutuku-works, and summarise it as Table 1 [7]

Descriptor	**Explanation**
prefLabel	Displays preferred terms in Māori. These are words and phrases chosen as subjects
altLabel	Displays preferred terms in English. These are words and phrases chosen as subjects
Tukutuku (TU)	Displays unpreferred terms in Māori. These are words and phrases not chosen as subjects
usedFor (UF)	Displays unpreferred terms in English. These are words and phrases not chosen as subjects
Whakamārama (WH)	Scope note in Māori describing the term
scopeNote	Scope note in English describing the term
Tāhuhu (TA)	Displays terms that have broader subject coverage
Heke (HE)	Displays terms that have narrower subject coverage
Kaho (KA)	Displays terms that are related in subject by may not form part of the same hierarchy

Table 1: Structure

With a basic understanding of the structure, we can easily spot 2 issues that need to be addressed in the original RDF/XML file as in Listing 2.1.

1. The IRI for a term as in the attribute *rdf:about* does not contain its corresponding Māori term. Though they can still be uniquely identified, they lose their meanings.
2. When a term is linked to other terms, other terms are represented as literals which cannot be effectively referenced.

```
<skos:Concept rdf:about="http://my.site.com/#Tohi%20m%E4taitai">
    <skos:prefLabel>Tohi m&#257;taitai</skos:prefLabel>
    <skos:scopeNote>The multitudes of Tangaroa
    ↪   &#40;foodstuffs&#41; gathered by people to eat. To
    ↪   some&#44; this word relates to shellfish found on
    ↪   rocks&#44; in the sand&#44; or marine life found
    ↪   inshore.</skos:scopeNote>
    <skos:WH>Ng&#257; tini a Tangaroa ka kohia hei kai m&#257;
    ↪   te tangata. Ki &#275;tahi&#44; ka wh&#257;iti mai te
    ↪   kupu nei ki ng&#257; hanga piri toka&#44; noho
    ↪   onep&#363;&#44; k&#257;ore e uru mai ng&#257; ika tere i
    ↪   te moana nui tonu.</skos:WH>
```

```
        <skos:altLabel>Gathering shellfish</skos:altLabel>
        <skos:UF>Shellfish gathering</skos:UF>
        <skos:TA>Mahinga ika t&#363;turu</skos:TA>
        <skos:KA>K&#257;kahi</skos:KA>
        <skos:STA>Approved</skos:STA>
        <skos:INP>2010-12-21</skos:INP>
        <skos:APP>2010-12-21</skos:APP>
        <skos:UPD>2011-06-03</skos:UPD>
        <skos:TNR>3399</skos:TNR>
    </skos:Concept>
```

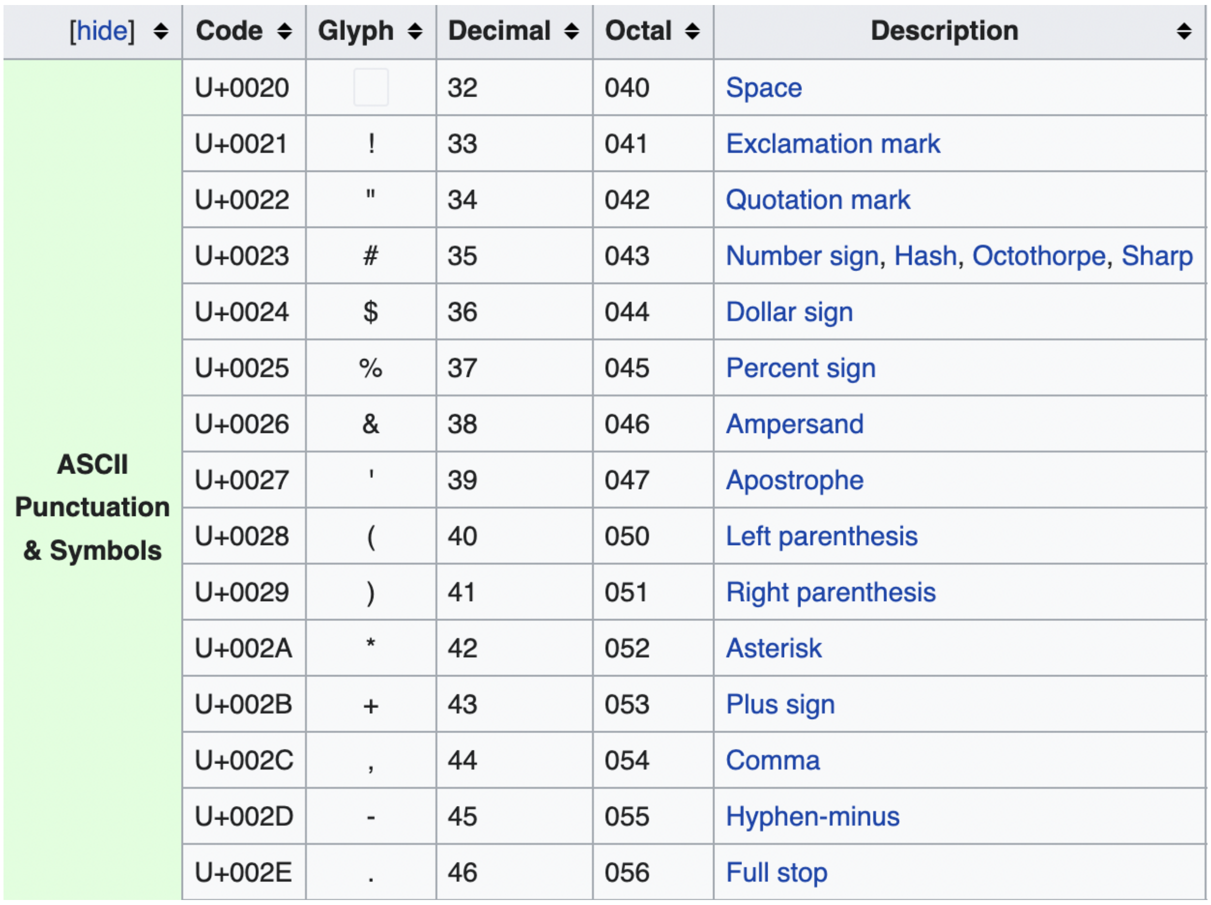
Listing 2.1: Thesaurus

2.3.1 Formatting URLs

The first issue can be addressed by attaching IRIs with meaningful localnames, and we want to use URL encoding to encode the textual content of the element *prefLabel*, which is the correctly formatted Māori term (CFMT).

Before we touch on URL encoding, we need to have some preliminary knowledge about Universal Character Set (Unicode) and Unicode Transformation 8 (UTF-8).

Unicode is an international standard for mapping characters used in natural language, mathematics, music, and other domains to machine-readable values. Those machine-readable values are called code points, and can be represented as a human-readable decimals, or further transformed into hexadecimal or octal numbers as shown in 2.4 [8].



[hide]	Code	Glyph	Decimal	Octal	Description
ASCII Punctuation & Symbols	U+0020		32	040	Space
	U+0021	!	33	041	Exclamation mark
	U+0022	"	34	042	Quotation mark
	U+0023	#	35	043	Number sign, Hash, Octothorpe, Sharp
	U+0024	$	36	044	Dollar sign
	U+0025	%	37	045	Percent sign
	U+0026	&	38	046	Ampersand
	U+0027	'	39	047	Apostrophe
	U+0028	(	40	050	Left parenthesis
	U+0029	)	41	051	Right parenthesis
	U+002A	*	42	052	Asterisk
	U+002B	+	43	053	Plus sign
	U+002C	,	44	054	Comma
	U+002D	-	45	055	Hyphen-minus
	U+002E	.	46	056	Full stop

Figure 2.4: Unicode

Then the code points from Unicode need to be further converted into binary, of which the process is called encoding. One of the most popular encoding scheme for Unicode is UTF-8, and UTF-8 works as follows [].

- For Unicode characters with code points from 0-127, they can be represented in a byte (there are 8 digits in a byte) with the first digit set to 0 (we can see that there are $2^{8-1} = 128$ combinations possible with remaining 7-digit binary)
- For Unicode characters with code points more than 128, the idea is to use the number of high-order 1's in the first byte to signal the number of bytes in the multi-byte character. Take the Latin letter é for example. It can be broken down into five logical components as shown in Figure 2.5 [9]

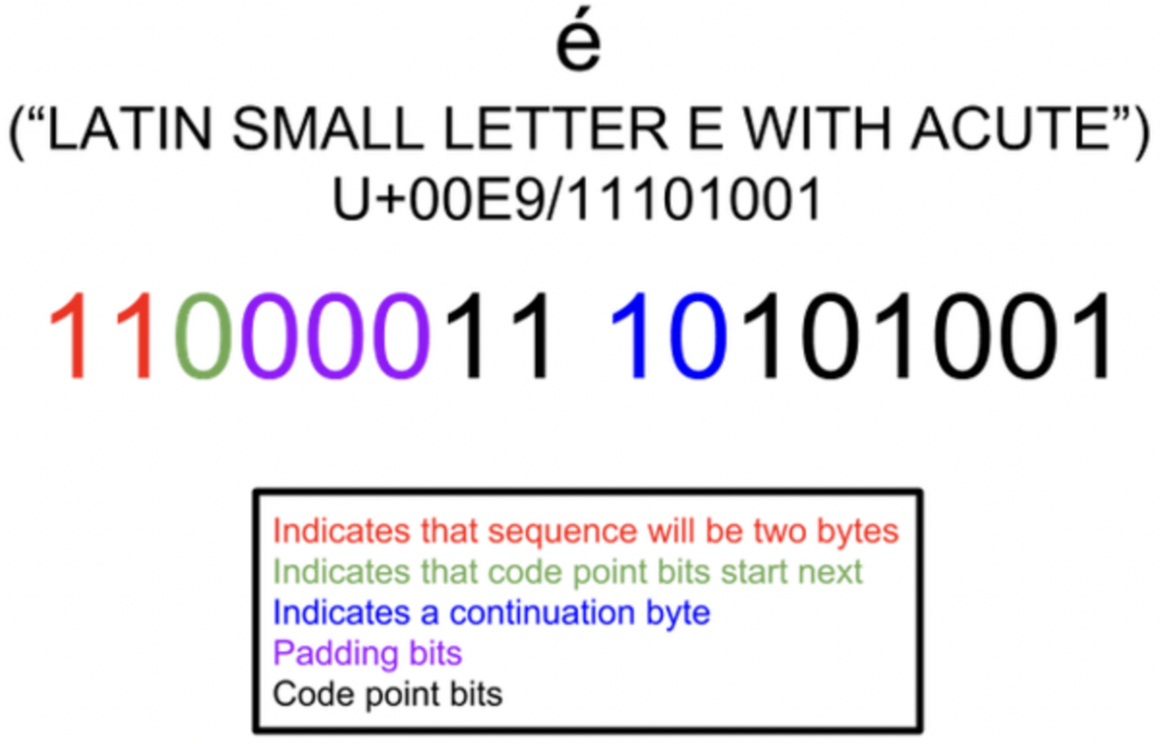

é
("LATIN SMALL LETTER E WITH ACUTE")
U+00E9/11101001
11000011 10101001
Indicates that sequence will be two bytes
Indicates that code point bits start next
Indicates a continuation byte
Padding bits
Code point bits

Figure 2.5: UTF-8

With the foundation of all above, URL encoding is simply to take the hexadecimal result of UTF-8 encoding, group it into 2-digit pairs, and give each pair a prefix % (that is why it is also called percent encoding) marking the start of a URL encoding. In that way, international letters can be encoded into URL.

Now, we check Listing 2.1 of our RDF/XML document, and can see that Tohi%20m%E4taitai is not a legal URL encoding as %E4 represents the byte 11100100 with the leading 1's saying it should be a 3-byte character, but it is not followed by any byte in the URL.

We can address the problem by URL encoding CFMT into the localname of an IRI. Then, the next question that begs for an answer is how do we extract localnames?

The answer is to use a regular expression. A regular expression is a sequence of characters that specifies a search pattern in text. Some common search patterns are
w for word character,
W for non-word character, ̂for the start of a string or line, $ for the end of a string or line, etc [10].

We know that localname starts with #, and we want to match it until the end of an IRI (which indicates a greedy mode where we keep matching until the search pattern can no longer be satisfied).

Thus, we can extract the localname of an IRI and then replace it with CFMT.

2.3.2 Referencing IRIs

The second issue that needs to be addressed is that when a term is linked to other terms by related (KA) / broader (TA) / narrower (HE), the literals are presented. That makes it hard for us to reference terms for literals are harder to locate than the unique IRIs. So we want to replace literals with the IRIs of the terms they are referring to.

We can achieve that by tracing a literal back to the equivalent CFMT, extracting the IRI for that CFMT, and replacing the literal with the extracted IRI.

2.3.3 Reuslt

Now, we combine the solutions we have discussed, we can write an XQuery as in Listing 2.2 to transform our original RDF/XML file for the thesaurus data.

```
let $terms := doc("./taxonomy.xml")
return
    <result>
    {
        for $term in $terms//*[skos:prefLabel]
        return
                <skos:Concept rdf:about="{
                 ↪  fn:replace($term/@rdf:about, "#[^$]+$", "#" ||
                 ↪  fn:encode-for-uri($term/skos:prefLabel)) }">
                    { $term/*[not(self::skos:HE or self::skos:TA or
 ↪  self::skos:KA)] }
                    {
                        for $broaderTerm in $term/skos:TA
                        let $temp := $terms//*[skos:prefLabel =
                         ↪  $broaderTerm]
                        return <skos:TA rdf:resource='{
                         ↪  fn:replace($temp/@rdf:about, "#[^$]+$",
                         ↪  "#" ||
                         ↪  fn:encode-for-uri($temp/skos:prefLabel))
                         ↪  }'> </skos:TA>
                    }
                    {
                        for $narrowerTerm in $term/skos:HE
                        let $temp := $terms//*[skos:prefLabel =
 ↪  $narrowerTerm]
                        return <skos:HE rdf:resource='{
 ↪  fn:replace($temp/@rdf:about, "#[^$]+$", "#" ||
 ↪  fn:encode-for-uri($temp/skos:prefLabel)) }'> </skos:HE>
                    }
                    {
                        for $relatedTerm in $term/skos:KA
                        let $temp := $terms//*[skos:prefLabel =
 ↪  $relatedTerm]
                        return <skos:KA rdf:resource='{
 ↪  fn:replace($temp/@rdf:about, "#[^$]+$", "#" ||
 ↪  fn:encode-for-uri($temp/skos:prefLabel)) }'> </skos:KA>
                    }
                </skos:Concept>
    }
    </result>
```

Listing 2.2: Thesaurus Data Transformation

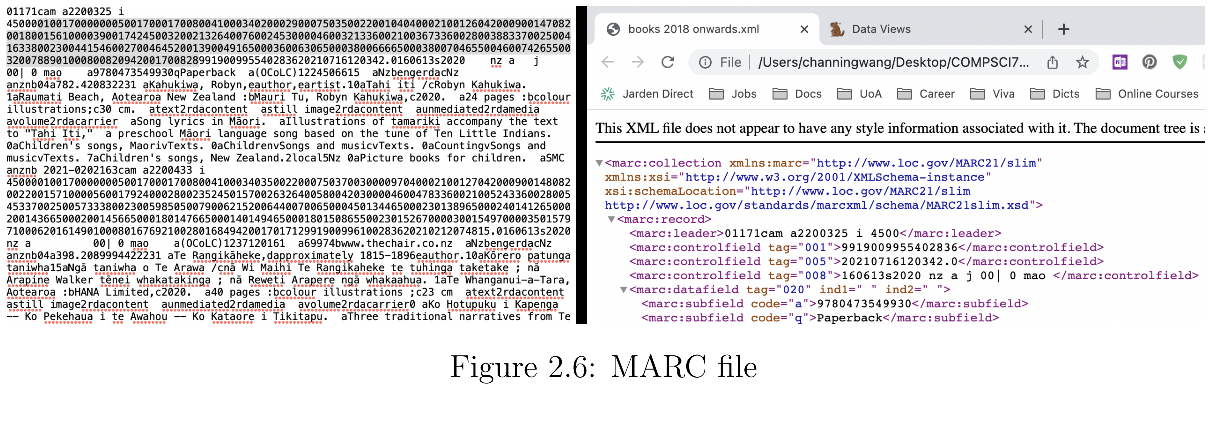
2.4 Bibliographic Data

2.4.1 Converting MARC Records

Bibliographic data comes in the format of MARC. MARC is a format commonly used by librarians. It follows the structure as mentioned below [11].

- Leader field to primarily provide information for the processing of the record
- Directory fields as a series of entries that contain the tag, length, and starting location of each variable field within a record
- Variable fields contain the data in a MARC bibliographic record, and each is identified by a three-character numeric tag that is stored in the Directory entry for the field
  - Variable control fields, The 00X fields
  - Variable data fields contain two indicator positions stored at the beginning of each field and a two-character subfield code preceding each data element within the field
    - Indicator positions - Indicator values are interpreted independently, that is, meaning is not ascribed to the two indicators taken together
    - Subfield codes are defined independently for each field; however, parallel meanings are preserved whenever possible (e.g., in the 100, 400, and 600 Personal Name fields)

In order to further process the data, we need to use MARCEdit to convert the MARC file into an MARC/XML document, and then simply by eyeballing the raw MARC file and the converted MARC/XML file side by side, we can notice directory fields (as highlighted in grey in Figure 2.6) are missing, but variable fields are retained in the converted file, so we assume there's no information loss during the process since we know that directory entries are only for the purpose of indexing the variable fields it entails.



```
01171cam a2200325 i
4500001001700000005001700017008004100034020002900075035002200104040002100126042000900147082
0018001561000039001742450032002132640076002453000046003213360021003673360028003883370025004
1633800230044154600270046452001390049165000360063065000380066665000380070465500460074265500
3200788901000800820942001700828991900995540283620210716120342.0160613s2020    nz a   j
00| 0 mao    a9780473549930qPaperback  a(OCoLC)1224506615  aNzbengerdacNz
anznb04a782.420832231 aKahukiwa, Robyn,eauthor,eartist.10aTahi iti /cRobyn Kahukiwa.
1aRaumati Beach, Aotearoa New Zealand :bMauri Tu, Robyn Kahukiwa,c2020.  a24 pages :bcolour
illustrations;c30 cm.  atext2rdacontent  astill image2rdacontent  aunmediated2rdamedia
avolume2rdacarrier  aSong lyrics in Māori.  aIllustrations of tamariki accompany the text
to "Tahi Iti,"  a preschool Māori language song based on the tune of Ten Little Indians.
0aChildren's songs, MaorivTexts. 0aChildrenvSongs and musicvTexts. 0aCountingvSongs and
musicvTexts. 7aChildren's songs, New Zealand.2local5Nz 0aPicture books for children.  aSMC
anznb 2021-0202163cam a2200433 i
4500001001700000005001700017008004100034035002200075037003000097040002100127042000900148082
0022001571000056001792400028002352450157002632640058004203000046004783360021005243360028005
4533700250057333800230059850500790062152006440070065000450134465000230138965000240141265000
2001436650002001456650001801476650001401494650001801508655002301526700003001549700003501579
7100062016149010008016769210028016849420017017129919009961002836202102120748l5.0160613s2020
nz a         00| 0 mao    a(OCoLC)1237120161  a69974bwww.thechair.co.nz  aNzbengerdacNz
anznb04a398.2089994422231 aTe Rangikāheke,dapproximately 1815-1896eauthor.10aKōrero patunga
taniwha15aNgā taniwha o Te Arawa /cnā Wi Maihi Te Rangikaheke te tuhinga taketake ; nā
Arapine Walker tēnei whakatakinga ; nā Reweti Arapere ngā whakaahua. 1aTe Whanganui-a-Tara,
Aotearoa :bHANA Limited,c2020.  a40 pages :bcolour illustrations ;c23 cm  atext2rdacontent
astill image2rdacontent  aunmediated2rdamedia  avolume2rdacarrier0 aKo Hotupuku i Kapenga
-- Ko Pekehaua i te Awahou -- Ko Kataore i Tikitapu.  aThree traditional narratives from Te
```

This XML file does not appear to have any style information associated with it. The document tree is

```
▼<marc:collection xmlns:marc="http://www.loc.gov/MARC21/slim"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://www.loc.gov/MARC21/slim
 http://www.loc.gov/standards/marcxml/schema/MARC21slim.xsd">
  ▼<marc:record>
     <marc:leader>01171cam a2200325 i 4500</marc:leader>
     <marc:controlfield tag="001">9919009955402836</marc:controlfield>
     <marc:controlfield tag="005">20210716120342.0</marc:controlfield>
     <marc:controlfield tag="008">160613s2020 nz a j 00| 0 mao </marc:controlfield>
   ▼<marc:datafield tag="020" ind1=" " ind2=" ">
       <marc:subfield code="a">9780473549930</marc:subfield>
       <marc:subfield code="q">Paperback</marc:subfield>
```

Figure 2.6: MARC file

2.4.2 Transforming MARC/XML

Now, we are going to investigate the bibliographic data containing mixed content of both physical and digital books, and we need to extract useful information from the XML file as shown in Listing 2.6, namely an Id, an International Standard Book Number (ISBN), a title, and download links.

- Id, the child of the element *marc:controlfield* with a tag attribute equal to *001*
- ISBN, the first child with a code attribute equal to *a* of the first *marc:datafield* with a tag attribute equal to *020* (for digital records, we only need the first ISBN to refer to ISBN dedicated to its digital copy)
- title - the concatenation of the children with a code attribute equal to *a* (title) and *b* (remainder) of the element *marc:datafield* with a tag attribute equal to *245*
- download links - the children with the code attribute equal to *u* of the element *marc:datafield* with a tag attribute equal to *856*

```
<marc:record>
<marc:leader>02185cam a2200421 i 4500</marc:leader>
<marc:controlfield tag="001">9919030266602836</marc:controlfield>
<marc:controlfield tag="005">20210203100523.0</marc:controlfield>
<marc:controlfield tag="006">m|||||o||d||||||||</marc:controlfield>
<marc:controlfield tag="007">cr#|||||||||||</marc:controlfield>
<marc:controlfield tag="008">201221s2020 nz a ob 00| 0 eng
↪  </marc:controlfield>
...
<marc:datafield tag="020" ind1=" " ind2=" ">
<marc:subfield code="a">9780947517236</marc:subfield>
<marc:subfield code="q">electronic</marc:subfield>
</marc:datafield>
...
<marc:datafield tag="020" ind1=" " ind2=" ">
<marc:subfield code="z">9780947517229</marc:subfield>
<marc:subfield code="q">print</marc:subfield>
</marc:datafield>
<marc:datafield tag="245" ind1="0" ind2="2">
<marc:subfield code="a">A review of the funding and prioritisation
↪  of environmental research in New Zealand.</marc:subfield>
</marc:datafield>
...
<marc:datafield tag="650" ind1=" " ind2="7">
<marc:subfield code="a">Waiata.</marc:subfield>
<marc:subfield code="2">reo</marc:subfield>
</marc:datafield>
<marc:datafield tag="650" ind1=" " ind2="7">
<marc:subfield code="a">Reorua.</marc:subfield>
<marc:subfield code="2">reo</marc:subfield>
</marc:datafield>
<marc:datafield tag="650" ind1=" " ind2="7">
```

```
<marc:subfield code="a">Waiata koroua.</marc:subfield>
<marc:subfield code="2">reo</marc:subfield>
</marc:datafield>
...
<marc:datafield tag="856" ind1="4" ind2="0">
<marc:subfield code="u">https://www.pce.parliament.nz/publications/e⌟
↪  nvironmental-research-funding-review</marc:subfield>
</marc:datafield>
<marc:datafield tag="856" ind1="4" ind2="0">
<marc:subfield
↪  code="u">https://natlib-primo.hosted.exlibrisgroup.com/primo-exp⌟
↪  lore/fulldisplay?docid=NLNZ_ALMA11356406180002836&context=L&vid=⌟
↪  NLNZ&search_scope=NLNZ&tab=catalogue&lang=en_US</marc:subfield>
...
```

Listing 2.3: MARC

2.4.3 Integrating with Thesaurus Data

That is just half the work done, and the most important part is to relate tagged terms from the bibliographic data with the term IRIs from thesaurus data.

Looking at Listing 2.3, we need children with a code attribute equal to a and the content of its sibling equal to *reo*. Considering the structure of our previous work, a similar approach can be used, that is, to trace each tagged term back to equivalent CFMT, extract the IRI for that CFMT, and replace the tagged term with the extracted IRI.

2.4.4 Result

Now, combine the solutions we have discussed, we can write an XQuery as in Listing 2.4 to transform our original MARC/XML file for the bibliographic data.

```
let $terms:=doc("./ontology.rdf")
let $records:=doc("./bibliography.xml")

return
    <result>
        {
            for $record in $records//marc:record
            return
            <record>
                {<recordId>{string($record/marc:controlfield[@tag="0⌟
↪  01"])}</recordId>}
                {
                    for $heading in $record/marc:datafield[@tag="650⌟
                    ↪  "]/marc:subfield[@code="2" and
                    ↪  text()="reo"]/preceding-sibling::marc:subfie⌟
                    ↪  ld[@code="a"]
```

```
           return <heading>{$terms//skos:Concept[(skos:pref⌟
            ↪  Label ||
            ↪  ".")=string($heading)]/@rdf:about}</heading>
         }
         {
           for $ISBN in
            ↪  $record/marc:datafield[@tag="020"][*/@code="⌟
            ↪  a"][*/@code="q"]/marc:subfield[@code="a"][1]
           return <ISBN>{$ISBN/text()}</ISBN>
         }
         {
           <Title>
           {
             let $temp :=
              ↪  $record/marc:datafield[@tag="245"]
             return string($temp/*[@code="a"] ||
              ↪  $temp/*[@code="b"])
           }
           </Title>}
         {
           for $Link in $record/marc:datafield[@tag="856"]/⌟
            ↪  *[@code="u"]
           return <Link>{string($Link)}</Link>
         }
       </record>
     }
   </result>
```

Listing 2.4: Bibliographic Data Transformation

3 Modelling Data

The graph database is naturally schemaless - you plug in data, and then it is ready to go. However, this kind of approach leaves a disconnection between business and technology. In a business context, you tend to create more data for a specific application, which is called operational data. That suits best in a quick iteration of software development but is created with little consideration of what data actually mean, which is called metadata. Over time, the gap between operational data and metadata will be wider unless a unified data architecture is established. Lessons taken, we will be doing data modelling up front for our ongoing project in the long run.

3.1 OWL

In RDF-based database, we use the ontology to provide definitions for resources and how they are related. There are many ontology languages to that purpose, and we will focus the one we chose - OWL. The only thing we have to remember is that we must stick to the syntax supported by OWL so that we can take advantage of its reasoning power at a later stage. OWL follows the syntax of RDF/XML document, with new vocabulary added to the mix.

3.1.1 Classes

Classes are the basic components of any ontology language to describe a category of things that have some property in common. Take Listing 3.1 for example [12].

- Classes and instances. The instance *Mary* is in the class *Woman*
- Class hierarchies
  - *Woman* is a class, and also a subclass of *Person*
  - *Person* is a class, and also an equivalent class of *Human*
- Class disjointness. The collection of *Woman* and *Man* is by definition a *AllDisjointClasses*

```
<Woman rdf:about="Mary"/>

<owl:Class rdf:about="Woman">
  <rdfs:subClassOf rdf:resource="Person"/>
</owl:Class>
<owl:Class rdf:about="Person">
   <owl:equivalentClass rdf:resource="Human"/>
 </owl:Class>

 <owl:AllDisjointClasses>
   <owl:members rdf:parseType="Collection">
     <owl:Class rdf:about="Woman"/>
     <owl:Class rdf:about="Man"/>
   </owl:members>
 </owl:AllDisjointClasses>
```

Listing 3.1: Classes

3.1.2 Properties

Like classes, properties are also very common. They are used to connect subjects to objects and always signify its subject (domain) and object (range). Take Listing 3.2 for example [12].

- Object properties
  - *John* is connected to *Mary* by the property *hasWife*
  - The assertion of *Bill* is connected to *Mary* by *hasWife* is by definition a *NegativePropertyAssertion*.
- Property hierarchies. *hasWife* is a sub-property of *hasSpouse*
- Domain and Range Restrictions. *HasWife* can be applied to the domain *Man* and the range *Woman*

```
<rdf:Description rdf:about="John">
  <hasWife rdf:resource="Mary"/>
</rdf:Description>
<owl:NegativePropertyAssertion>
  <owl:sourceIndividual rdf:resource="Bill"/>
  <owl:assertionProperty rdf:resource="hasWife"/>
  <owl:targetIndividual rdf:resource="Mary"/>
</owl:NegativePropertyAssertion>

<owl:ObjectProperty rdf:about="hasWife">
  <rdfs:subPropertyOf rdf:resource="hasSpouse"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:about="hasWife">
  <rdfs:domain rdf:resource="Man"/>
  <rdfs:range rdf:resource="Woman"/>
</owl:ObjectProperty>
```

Listing 3.2: Properties

3.1.3 Individuals

As OWL 2 is based on an open world assumption, which means that it is not assuming what it does not know to be false, people with 2 different names can be the same person, the pet can be owned by multiple people, just to name a few. Therefore, it is important that there is a mechanism to assert facts explicitly about individuals to make some clarifications in a business context. Take Listing 3.3 for example [12].

- Equality and inequality of individuals. *John* is a different individual from *Bill*, but *Jim* is the same individual as *James*.
- Datatypes. The target value of the *NegativePropertyAssertion* is of data type integer.

```
<rdf:Description rdf:about="John">
  <owl:differentFrom rdf:resource="Bill"/>
</rdf:Description>
<rdf:Description rdf:about="James">
  <owl:sameAs rdf:resource="Jim"/>
</rdf:Description>

<Person rdf:about="John">
  <hasAge rdf:datatype="http://www.w3.org/2001/XMLSchema#integer">⌟
  ↪  51</hasAge>
</Person>
<owl:NegativePropertyAssertion>
  <owl:sourceIndividual rdf:resource="Jack"/>
  <owl:assertionProperty rdf:resource="hasAge"/>
```

```
    <owl:targetValue
     ↪  rdf:datatype="http://www.w3.org/2001/XMLSchema#integer">
      53
    </owl:targetValue>
  </owl:NegativePropertyAssertion>
```

Listing 3.3: Individuals

3.1.4 Advanced

There are quite a few advanced features of OWL 2, but we only touch on the ones that will be used for our data modelling. Also, since they are more complicate than what we mentioned before, first order logic (FOL) is introduced to make a better understanding.

- Q is the inverse of the property P means $\forall x, y.P(x, y) \iff Q(y, x)$. Note that the inverse property is a generalisation of the symmetric property where the direction of a property does not matter and can be represented as $\forall x, y.P(x, y) \iff P(y, x)$. Take Listing 3.4 for example, *hasParent* is the inverse property of *hasChild*, while *hasSpouse* is a symmetric property working both ways.
- The transitivity of a property P means $\forall x, y, z.(P(x, y) \wedge P(y, z)) \rightarrow P(x, z)$. Take Listing 3.4 for example, *hasAncestor* is a transitive property.

```
<owl:ObjectProperty rdf:about="hasParent">
  <owl:inverseOf rdf:resource="hasChild"/>
</owl:ObjectProperty>
<owl:SymmetricProperty rdf:about="hasSpouse"/>

 <owl:TransitiveProperty rdf:about="hasAncestor"/>
```

Listing 3.4: Advanced

3.2 Ontology

3.2.1 Analysing Data

Now that we have learnt OWL, it is time to use this ontology language to model our data. However, without a better understanding of the data itself, the model is most likely flawed. So, we will analyse our data first.

In the Māori worldview, aspects of taha tinana/the people, taha wairua/the spiritual and taha hinengaro/the mind are intrinsically connected and related to each other. This model recognises both the traditional and contemporary perspectives as shown in Figure 3.1 [13].

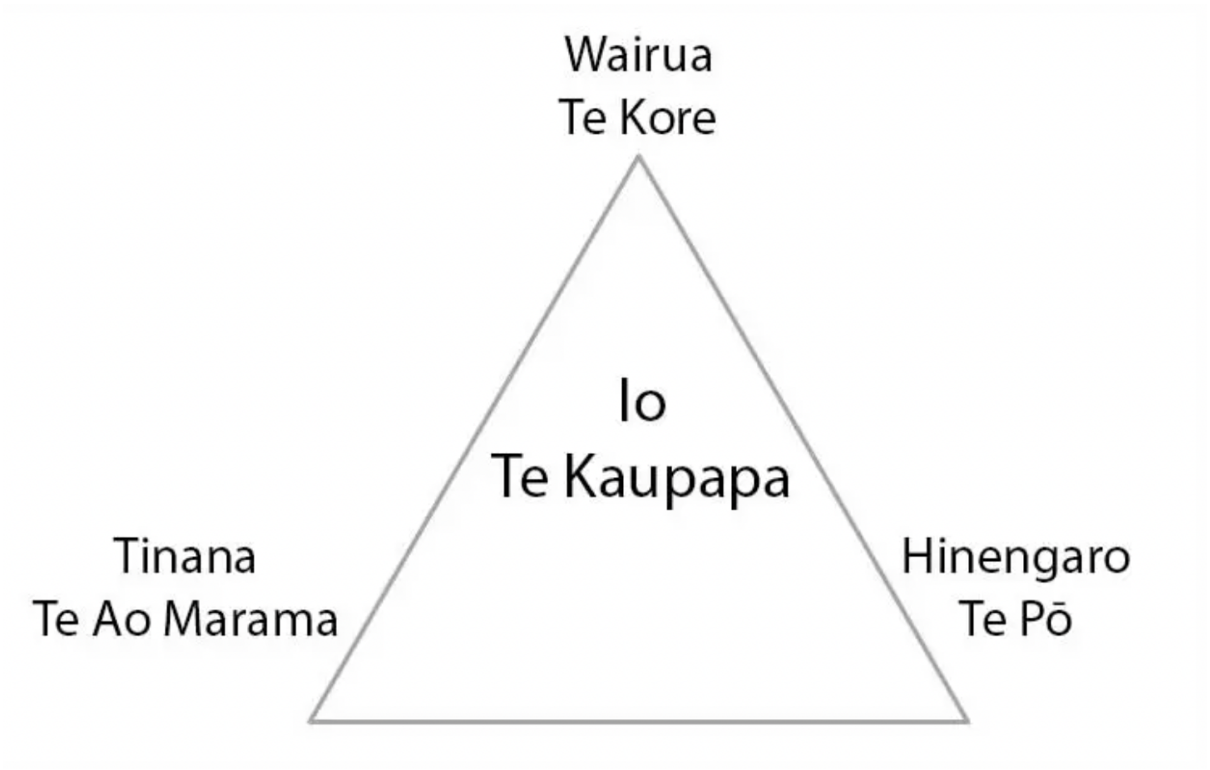


Figure 3.1: Conceptual framework

Metaphorically, those top concepts are the load bearing posts to hold the roof. Then, we can compare how terms are connected to ridges running down the beam of a roof, the adjacent related terms can be going both ways, while broader / narrower terms are going one way from top down /from bottom up as shown in Figure 3.2.

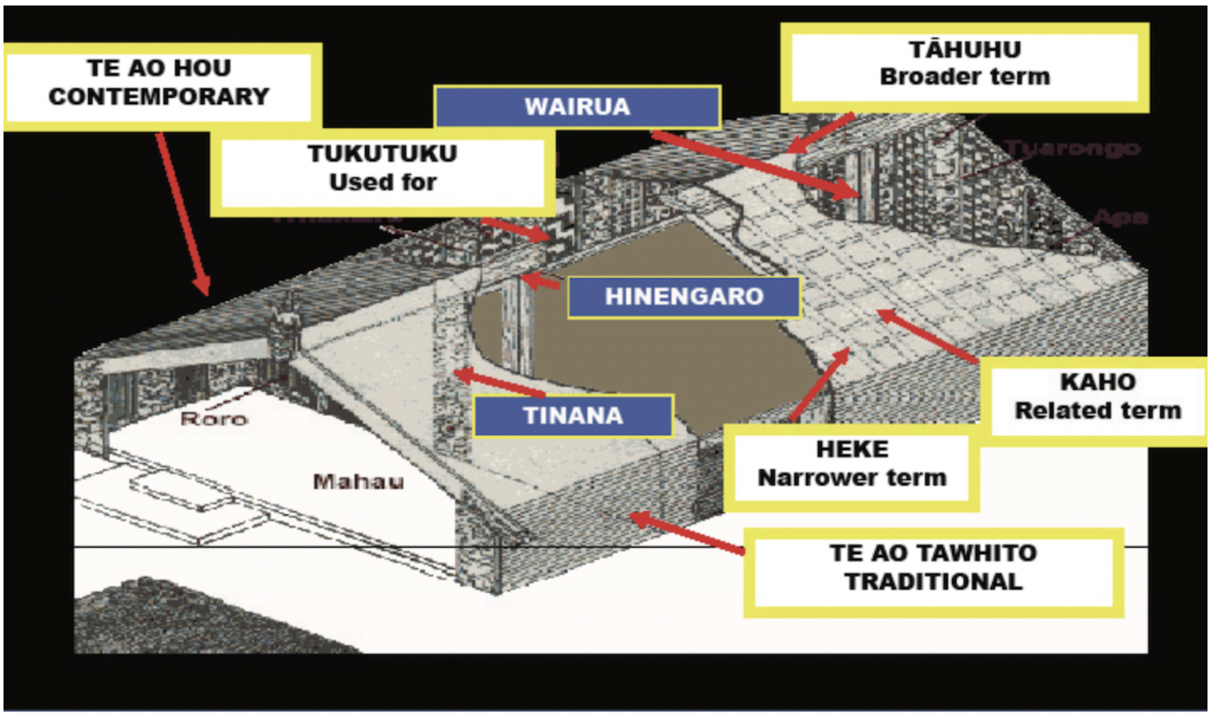


Figure 3.2: Architect

Therefore, broader (TA) and narrower (HE) are transitive properties, while related (KA) is a symmetric property.

3.2.2 Result

Now that we figure out how terms are related with regards to ontology, we can write a RDF/XML document as in Listing 3.5. Thus, we can go in the direction of the transitivity from the immediate broader / narrower terms to the further broader / narrower terms and jump between related terms by taking the power of the built-in reasoner to make inferences for us.

```
<owl:ObjectProperty
 ↪  rdf:about="http://www.w3.org/2008/05/skos#semanticRelation">
    <rdfs:domain
     ↪  rdf:resource="http://www.w3.org/2008/05/skos#Concept" />
    <rdfs:range
     ↪  rdf:resource="http://www.w3.org/2008/05/skos#Concept" />
</owl:ObjectProperty>
<owl:ObjectProperty
 ↪  rdf:about="http://www.w3.org/2008/05/skos#broader">
    <rdfs:subPropertyOf rdf:resource="http://www.w3.org/2008/05/⌋
     ↪  skos#broaderTransitive"
     ↪  />
    <owl:inverseOf
     ↪  rdf:resource="http://www.w3.org/2008/05/skos#narrower" />
</owl:ObjectProperty>
<owl:ObjectProperty
 ↪  rdf:about="http://www.w3.org/2008/05/skos#narrower">
    <rdfs:subPropertyOf rdf:resource="http://www.w3.org/2008/05/⌋
     ↪  skos#narrowerTransitive"
     ↪  />
    <owl:inverseOf
     ↪  rdf:resource="http://www.w3.org/2008/05/skos#broader" />
</owl:ObjectProperty>
<owl:ObjectProperty
 ↪  rdf:about="http://www.w3.org/2008/05/skos#related">
    <rdfs:subPropertyOf rdf:resource="http://www.w3.org/2008/05/⌋
     ↪  skos#semanticRelation"
     ↪  />
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#Symmet⌋
     ↪  ricProperty"
     ↪  />
</owl:ObjectProperty>
<owl:ObjectProperty
 ↪  rdf:about="http://www.w3.org/2008/05/skos#broaderTransitive">
    <rdfs:subPropertyOf rdf:resource="http://www.w3.org/2008/05/⌋
     ↪  skos#semanticRelation"
     ↪  />
```

```
        <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#Transi
        ↪  tiveProperty"
        ↪  />
        <owl:inverseOf rdf:resource="http://www.w3.org/2008/05/skos#
        ↪  narrowerTransitive"
        ↪  />
    </owl:ObjectProperty>
    <owl:ObjectProperty rdf:about="http://www.w3.org/2008/05/skos#na
    ↪  rrowerTransitive">
        <rdfs:subPropertyOf rdf:resource="http://www.w3.org/2008/05/
        ↪  skos#semanticRelation"
        ↪  />
        <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#Transi
        ↪  tiveProperty"
        ↪  />
        <owl:inverseOf rdf:resource="http://www.w3.org/2008/05/skos#
        ↪  broaderTransitive"
        ↪  />
    </owl:ObjectProperty>
```

Listing 3.5: Ontology

4 Accessing Data

Since we are using SPARQL to query data, we need to first know the SPARQL query language.

4.1 SPARQL

As we can see that RDF is a directed, single-labelled graph, SPARQL is a query language designated for that kind of graph.

4.1.1 Graph Patterns

SPARQL is based on graph pattern matching. More complex graph patterns can be formed by combining smaller patterns in various ways, as we discuss below [14].

Basic Graph Patterns and *Group Graph pattern* are paired together for discussion. They provide a solution where a set of triple patterns are matched in a RDF graph. A triple pattern is made of a triple where any of subject, predicate or object can be replaced with a variable denoted by `?<variable name>`. You can use the delimiters to group them into different logical components - if you do, it is a group graph pattern, if you don't, it is a basic graph pattern. Mind that the outer-most graph pattern in a query is called the query pattern that only can be identified by the group delimiter . Take Listing 4.1 for example. The two queries yield the same solution, but they have different structures. The first one is made of one basic graph pattern, while the second one is made of two group patterns, and in each group pattern there is one basic graph pattern.

```
PREFIX foaf:    <http://xmlns.com/foaf/0.1/>
SELECT ?name ?mbox
WHERE  {
          ?x foaf:name ?name .
          ?x foaf:mbox ?mbox .
       }

PREFIX foaf:    <http://xmlns.com/foaf/0.1/>
SELECT ?name ?mbox
WHERE  { { ?x foaf:name ?name . }
         { ?x foaf:mbox ?mbox . }
       }
```

Listing 4.1: Basic and Group Graph Patterns

Then, we continue to pair up *optional Graph patterns* and *alternative Graph Pattern* for discussion, as we can be confused by these two due to their similarities. Optional graph pattern does not assume a complete graph pattern, so when the optional parts do not match, it creates no binding but does not discard the solution, while the alternative graph pattern assumes a complete graph pattern but then combine the results. Take Listing 4.2 for example, optional query yield a solution where Bob is included even if he has no *mbox* at all, while alternative query yields a solution where a collection of titles are matched even if they are of different versions of *dc*.

```
# Prefix
@prefix foaf:       <http://xmlns.com/foaf/0.1/> .
@prefix foaf:       <http://xmlns.com/foaf/0.1/> .
@prefix rdf:        <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .

# Data
_:a  rdf:type        foaf:Person .
_:a  foaf:name       "Alice" .
_:a  foaf:mbox      <mailto:alice@example.com> .
_:a  foaf:mbox       <mailto:alice@work.example> .

_:b  rdf:type        foaf:Person .
_:b  foaf:name       "Bob" .

# Optional Query
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name ?mbox
WHERE  { ?x foaf:name  ?name .
         OPTIONAL { ?x  foaf:mbox  ?mbox }
       }
# Prefix
@prefix dc10:  <http://purl.org/dc/elements/1.0/> .
@prefix dc11:  <http://purl.org/dc/elements/1.1/> .
```

```
# Data
_:a  dc10:title     "SPARQL Query Language Tutorial" .
_:a  dc10:creator   "Alice" .

_:b  dc11:title     "SPARQL Protocol Tutorial" .
_:b  dc11:creator   "Bob" .

_:c  dc10:title     "SPARQL" .
_:c  dc11:title     "SPARQL (updated)" .

# Alternative Query
SELECT ?title
WHERE  { { ?book dc10:title  ?title } UNION { ?book dc11:title
 ↪  ?title } }
```

Listing 4.2: Optional and Alternative Graph Patterns

Lastly, *Patterns on Named Graphs* is an extension to support the named graph in RDF so the query can be scoped to a specific named graph rather than the whole database.

4.1.2 Restrictions

After investigating graph patterns, we consider to add some restrictions so the query can be more precise. We can not get to all of them, so we will only talk about the ones used in our project as we shall see below.

The first is to put some restrictions on variables. This is expressed by the keyword *Filter* to control the whole group where it appears. Take Listing 4.3 for example, the three queries have the same constraint on the whole group despite the different positions of the keyword *Filter*

```
 SELECT ?x where
 {  ?x foaf:name ?name .
    ?x foaf:mbox ?mbox .
    Filter regex(?name, "Smith")
 }

 SELECT ?x where
 {  Filter regex(?name, "Smith")
    ?x foaf:name ?name .
    ?x foaf:mbox ?mbox .
 }

 SELECT ?x where
 {  ?x foaf:name ?name .
    Filter regex(?name, "Smith")
    ?x foaf:mbox ?mbox .
```

```
}
```

Listing 4.3: Filters

The second and the last one to mention is the keyword *Distinct* to ensure the sequences in an entire solution are unique, that is, if a sequence appears multiple times in the solution, only one of them will be kept.

4.2 Web Stack

It is not realistic to ask non-IT professionals to use SPARQL to query a RDF database. Therefore, a web application is proposed to address that.

4.2.1 Web UI

There are two main approaches to build web UI [15].

- Apps that render UI from the server, that is, when a request is sent to the server, server will handle the request, generate a page, and send it back to the client.
- Apps that render UI on the client in the browser, that is, the server is hosted in the same browser environment that the client is running so that everything can be run locally within a browser and get immediate response.

In our case, there are not too much interactivity happening within the client itself, everything is an intermediate layer to relay the query request to the database, so a server-rendered web UI is the best choice.

4.2.2 Service Lifetime

For a server-rendered web app, services are the additional features provided by the server that the app can use. One important thing is when the services are configured, there are three options to control their lifetime [16].

- Per request (transient)
- Per connection (scoped)
- Per instance (singleton)

As every client is going to maintain a connection with the server during a session, it is safe to think that a client will not be in conflict with other clients in a scoped lifetime.

4.2.3 HTTP Protocols

As graph database is relatively new, the only option we have to send requests to the database and get responses from the database is HTTP protocol [17].

For a request.

- Address, in the format of `<Host IP>/<catalogue>/<repository>`
- Type, GET / POST
- Query, URL encoded SPARQL query expression

- Headers
  - Authorisation, in the format of `<username>:<password> (Base64 encoded)`
  - Accept, *application/json*

For a response, we can extract the content and parse the encapsulated JSON string.

4.3 Features

Additional features are added as follows.

- The user name and password are communicated between the web app and the database. We put this kind of information in a separated configuration file so it is not shared.
- As the web app is open to the general public, a cloud-ready solution is created with *GitHub Action* to control the workflow of automatic continuous development (CI) and continuous integration (CD)

4.3.1 Querying Database

The basic functionality are to search, assemble the results for a specific term and browse the transitive broader / narrower terms. We will be going over those scenarios one by one.

The search is to check if the search string is present in an original Māori term or a translated Enligsh term. We need to check preferred and unpreferred uses and combine the results. We can write SPARQL expression as in Listing 4.4.

```
SELECT distinct ?s where
{
                { ?s <http://www.w3.org/2008/05/skos#prefLabel> ?o.
                 ↪ FILTER (contains(lcase(str(?o)), <search
                 ↪ string>)) }
                union { ?s <http://www.w3.org/2008/05/skos#usedFor>
                 ↪ ?o. FILTER (contains(lcase(str(?o)), <search
                 ↪ string>)) }
                union { ?s <http://www.w3.org/2008/05/skos#altLabel>
                 ↪ ?o. FILTER (contains(lcase(str(?o)),<search
                 ↪ string>)) }
                union { ?s <http://www.w3.org/2008/05/skos#tukutuku>
                 ↪ ?o. FILTER (contains(lcase(str(?o)), <search
                 ↪ string>)) }
}
```

Listing 4.4: Search

To assemble the results for a specific term is to gather all the information from a term with respect to its IRI, and then hand it over to the web app for the presentation. We can write SPARQL expression as in Listing 4.5.

```
SELECT distinct ?p ?o {<term IRI> ?p ?o }
```

Listing 4.5: Result

To browse the transitive broader / narrower terms is to query into the entailed triples inferred by our ontology. We can write SPARQL expression as in Listing 4.6

```
SELECT distinct ?o {<term IRI>
 ↪  <http://www.w3.org/2008/05/skos#broaderTransitive> ?o }
```

Listing 4.6: Transitivity

4.3.2 Developing Web Application

For starter, we will register a service to centralise the creation of the client to handle HTTP protocol for us, and we can write code as in Listing 4.7.

```
builder.Services.AddHttpClient("AllegroGraph", httpClient =>
{
    httpClient.BaseAddress = new Uri(builder.Configuration["db"]);
    httpClient.DefaultRequestHeaders.Accept.Add(new
     ↪  MediaTypeWithQualityHeaderValue("application/json"));
    httpClient.DefaultRequestHeaders.Authorization = new
     ↪  AuthenticationHeaderValue(
        "Basic",
        Convert.ToBase64String(System.Text.ASCIIEncoding.ASCII.GetBy↵
         ↪  tes(string.Format("{0}:{1}",
         ↪  builder.Configuration["usr"],
         ↪  builder.Configuration["pwd"]))));
});
```

Listing 4.7: HTTP Client

Then, integrating queries into the web app, take Listing 4.8 for example to implement the search component.

```
        public async Task<IActionResult> OnPostAsync()
        {
            if (!ModelState.IsValid) return Page();

            HttpClient client =
             ↪  _httpClientFactory.CreateClient("AllegroGraph");

            StringBuilder uri = new StringBuilder();
            uri.Append(_configuration["repository"]);
            string query = string.Format(
                @"SELECT distinct ?s {{
```

```
                {{ ?s <http://www.w3.org/2008/05/skos#prefLabel> ?o.
↪ FILTER (contains(lcase(str(?o)),'{0}')) }}
                union {{ ?s <http://www.w3.org/2008/05/skos#usedFor>
↪ ?o. FILTER (contains(lcase(str(?o)),'{0}')) }}
                union {{ ?s
↪ <http://www.w3.org/2008/05/skos#altLabel> ?o. FILTER
↪ (contains(lcase(str(?o)),'{0}')) }}
                union {{ ?s
↪ <http://www.w3.org/2008/05/skos#tukutuku> ?o. FILTER
↪ (contains(lcase(str(?o)),'{0}')) }}}}",
↪ searchString?.ToLower().Trim());
            uri.Append(HttpUtility.UrlEncode(query));
            HttpResponseMessage response = await
             ↪ client.GetAsync(uri.ToString());

            var result =
             ↪ JsonSerializer.Deserialize<Helper.AllegroGraphJsonRe⌟
             ↪ sult>(response.Content.ReadAsStringAsync().Result);
            List<string> iris = new List<string>();
            foreach (var match in result.values)
            {
                iris.Add(match[0]);
            }


            Helper.result[Helper.searchResult].Clear();
            Helper.result[Helper.searchResult].AddRange(iris);

            return RedirectToPage("Result", new { key =
             ↪ Helper.searchResult });
        }
```

Listing 4.8: Search on Web App

4.3.3 Result

You are welcome to check out Māori Subject Heading to play around the web app.

5 Conclusion

The project has been a journey of discovery, from knowing little even about graph databases to developing a functioning web app. To make the project work, there were lots of trials and errors, such as XQuery scripts that returned empty solutions, the reasoner that did not make inferences, and the web app requests that did not get sent through to the database. The whole process of this rinse-and-repeat of making assumptions, testing hypotheses, failing hypotheses until it worked is itself rewarding. I think, it also gives this ongoing project a foundation to build on. As the quote goes, *premature optimisation* *is the root of all evil.* Now, it is time to bring on the optimisation. I am open to all critics about my part of the project because it cannot be perfect, and it will never be, and that is the whole point for me to keep trying.

6 Future Work

I will start criticising myself by the things that I could have done better as below.

- XQeury scripts that can be rewritten in XSLT because it is a better language for transformation
- A data model that models the bibliographic data so that a record tagged by a term can also be tagged by its broader terms
- An API for authorised users to import their bibliographic and thesaurus data, and automatically generate recommended tags if there are download links providing access to the digital copies for analysis

Now, it is your turn.

References

[1] R. East, “Indigenous subject access: Nga upoko tukutuku,” *ANZTLA EJournal*, no. 1, pp. 37–67, 2008.

[2] “Xml.” [Online]. Available: https://www.w3.org/XML/

[3] “Xpath.” [Online]. Available: https://www.w3.org/TR/1999/REC-xpath-19991116/

[4] “Xquery.” [Online]. Available: https://www.w3.org/TR/xquery-31/

[5] “Rdf concept.” [Online]. Available: https://www.w3.org/TR/rdf11-concepts/

[6] “Rdf 1.1 xml syntax.” [Online]. Available: https://www.w3.org/TR/rdf-syntax-grammar/

[7] “how-nga-upoko-tukutuku-works.” [Online]. Available: https://natlib.govt.nz/librarians/nga-upoko-tukutuku/how-nga-upoko-tukutuku-works

[8] “Unicode.” [Online]. Available: https://en.wikipedia.org/wiki/Unicode

[9] “Utf-8.” [Online]. Available: http://doktermantul.com/

[10] “Regular expression.” [Online]. Available: https://en.wikipedia.org/wiki/Regular_expression

[11] “Marc 21 format for bibliographic data.” [Online]. Available: https://www.loc.gov/marc/bibliographic/

[12] “Owl 2.” [Online]. Available: https://www.w3.org/TR/owl2-primer/

[13] “nga-upoko-tukutuku.” [Online]. Available: https://natlib.govt.nz/librarians/nga-upoko-tukutuku

[14] “Sparql 1.1 query language.” [Online]. Available: https://www.w3.org/TR/sparql11-query/

[15] “Web ui.” [Online]. Available: https://docs.microsoft.com/en-us/aspnet/core/tutorials/choose-web-ui?view=aspnetcore-6.0

[16] “fundamentals.” [Online]. Available: https://docs.microsoft.com/en-us/aspnet/core/fundamentals/?view=aspnetcore-6.0&tabs=macos

[17] “Http protocol.” [Online]. Available: https://franz.com/agraph/support/documentation/current/http-protocol.html

List of Figures

List of Tables

List of Listings