Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Autonomous Systems Lab
Prof. Roland Siegwart

Master's Thesis

Brel

Autumn Term 2023

Supervised by:
Ghislain Fourny
Gustavo Alonso

Author:
Robin Schmidiger

Declaration of Originality

I hereby declare that the written work I have submitted entitled

Your Project Title

is original work which I alone have authored and which is written in my own words.[1]

Author(s)

First name	Last name

Student supervisor(s)

First name	Last name

Supervising lecturer

Roland	Siegwart

With the signature I declare that I have been informed regarding normal academic citation rules and that I have read and understood the information on 'Citation etiquette' (`https://www.ethz.ch/content/dam/ethz/main/education/rechtliches-abschluesse/leistungskontrollen/plagiarism-citationetiquette.pdf`). The citation conventions usual to the discipline in question here have been respected.

The above written work may be tested electronically for plagiarism.

Place and date   Signature

[1]Co-authored work: The signatures of all authors are required. Each signature attests to the originality of the entire piece of written work in its final form.

Contents

Preface

Bla bla ...

Abstract

Hier kommt der Abstact hin ...

Symbols

Symbols

ϕ, θ, ψ	roll, pitch and yaw angle
b	gyroscope bias
Ω_m	3-axis gyroscope measurement

Indices

x	x axis
y	y axis

Acronyms and Abbreviations

ETH	Eidgenössische Technische Hochschule
EKF	Extended Kalman Filter
IMU	Inertial Measurement Unit
UAV	Unmanned Aerial Vehicle
UKF	Unscented Kalman Filter

Chapter 1

Introduction

1.1 Motivation

Chapter 2

Einige wichtige Hinweise zum Arbeiten mit LATEX

Nachfolgend wird die Codierung einiger oft verwendeten Elemente kurz beschrieben. Das Einbinden von Bildern ist in LATEX nicht ganz unproblematisch und hängt auch stark vom verwendeten Compiler ab. Typisches Format für Bilder in LATEX ist EPS[1] oder PDF[2].

2.1 Gliederungen

Ein Text kann mit den Befehlen `\chapter{.}`, `\section{.}`, `\subsection{.}` und `\subsubsection{.}` gegliedert werden.

2.2 Referenzen und Verweise

Literaturreferenzen werden mit dem Befehl `\citep{.}` und `\citet{.}` erzeugt. Beispiele: ein Buch [6], ein Buch und ein Journal Paper [6, 4], ein Konferenz Paper mit Erwähnung des Autors: **(author?)** [5].
Zur Erzeugung von Fussnoten wird der Befehl `\footnote{.}` verwendet. Auch hier ein Beispiel[3].
Querverweise im Text werden mit `\label{.}` verankert und mit `\cref{.}` erzeugt. Beispiel einer Referenz auf das zweite Kapitel: chapter 2.

2.3 Aufzählungen

Folgendes Beispiel einer Aufzählung ohne Numerierung,

- Punkt 1
- Punkt 2

wurde erzeugt mit:

```
\begin{itemize}
  \item Punkt 1
  \item Punkt 2
\end{itemize}
```

[1] Encapsulated Postscript
[2] Portable Document Format
[3] Bla bla.

Folgendes Beispiel einer Aufzählung mit Numerierung,

1. Punkt 1
2. Punkt 2

wurde erzeugt mit:

```
\begin{enumerate}
  \item Punkt 1
  \item Punkt 2
\end{enumerate}
```

Folgendes Beispiel einer Auflistung,

P1 Punkt 1

P2 Punkt 2

wurde erzeugt mit:

```
\begin{description}
  \item[P1] Punkt 1
  \item[P2] Punkt 2
\end{description}
```

2.4 Erstellen einer Tabelle

Ein Beispiel einer Tabelle:

Table 2.1: Daten der Fahrzyklen ECE, EUDC, NEFZ.

Kennzahl	Einheit	ECE	EUDC	NEFZ
Dauer	s	780	400	1180
Distanz	km	4.052	6.955	11.007
Durchschnittsgeschwindigkeit	km/h	18.7	62.6	33.6
Leerlaufanteil	%	36	10	27

Die Tabelle wurde erzeugt mit:

```
\begin{table}[h]
\begin{center}
 \caption{Daten der Fahrzyklen ECE, EUDC, NEFZ.}\vspace{1ex}
 \label{tab:tabnefz}
 \begin{tabular}{ll|ccc}
 \hline
 Kennzahl & Einheit & ECE & EUDC & NEFZ \\ \hline \hline
 Dauer & s & 780 & 400 & 1180 \\
 Distanz & km & 4.052 & 6.955 & 11.007 \\
 Durchschnittsgeschwindigkeit & km/h & 18.7 &  62.6 & 33.6 \\
 Leerlaufanteil & \% & 36 & 10 & 27 \\
 \hline
 \end{tabular}
\end{center}
\end{table}
```

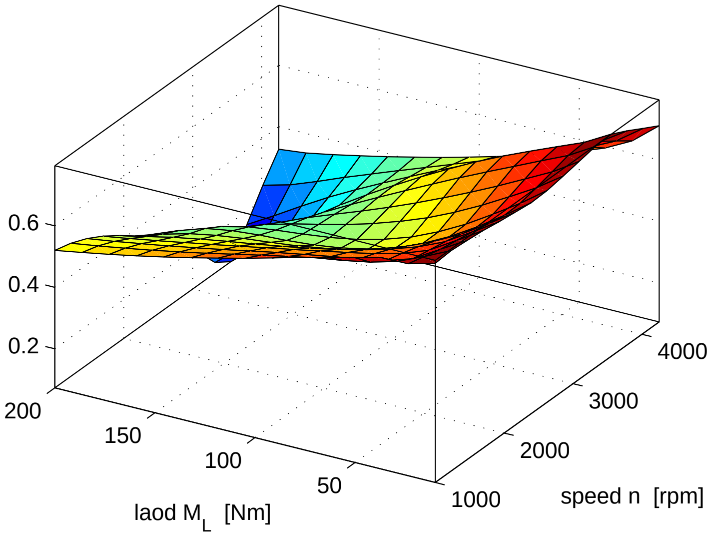

Figure 2.1: Ein Bild

2.5 Einbinden einer Grafik

Das Einbinden von Graphiken kann wie folgt bewerkstelligt werden:

```
\begin{figure}
   \centering
   \includegraphics[width=0.75\textwidth]{images/k_surf.pdf}
   \caption{Ein Bild.}
   \label{fig:k_surf}
\end{figure}
```

oder bei zwei Bildern nebeneinander mit:

```
\begin{figure}
  \begin{minipage}[t]{0.48\textwidth}
    \includegraphics[width = \textwidth]{images/cycle_we.pdf}
  \end{minipage}
  \hfill
  \begin{minipage}[t]{0.48\textwidth}
    \includegraphics[width = \textwidth]{images/cycle_ml.pdf}
  \end{minipage}
  \caption{Zwei Bilder nebeneinander.}
  \label{pics:cycle}
\end{figure}
```

2.6 Mathematische Formeln

Einfache mathematische Formeln werden mit der equation-Umgebung erzeugt:

$$p_{me0f}(T_e, \omega_e) \;=\; k_1(T_e) \cdot (k_2 + k_3 S^2 \omega_e^2) \cdot \mathit{\Pi}_{\max} \cdot \sqrt{\frac{k_4}{B}}\,. \tag{2.1}$$

Der Code dazu lautet:

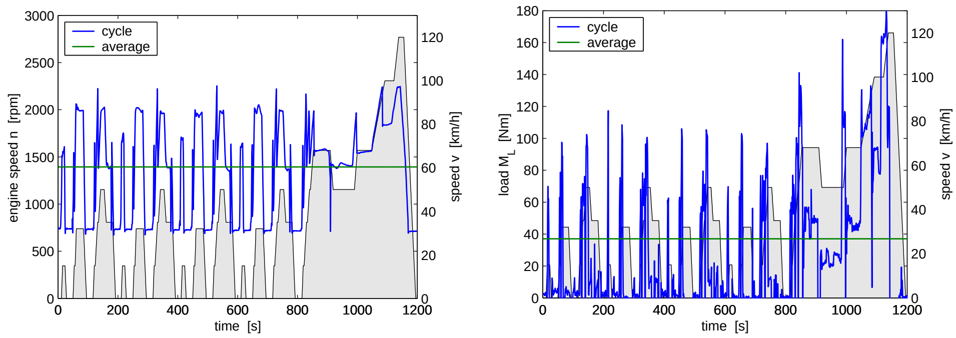

Figure 2.2: Zwei Bilder nebeneinander

```
\begin{equation}
 p_{me0f}(T_e,\omega_e) \ = \ k_1(T_e) \cdot (k_2+k_3 S^2
 \omega_e^2) \cdot \Pi_{max} \cdot \sqrt{\frac{k_4}{B}} \, .
\end{equation}
```

Mathematische Ausdrücke im Text werden mit \$formel\$ erzeugt (z.B.: $a^2+b^2=c^2$). Vektoren und Matrizen werden mit den Befehlen `\vec{.}` und `\mat{.}` erzeugt (z.B. $\boldsymbol{v}$, $\boldsymbol{M}$).

2.7 Weitere nützliche Befehle

Hervorhebungen im Text sehen so aus: *hervorgehoben*. Erzeugt werden sie mit dem `\epmh{.}` Befehl.
Einheiten werden mit den Befehlen `\unit[1]{m}` (z.B. 1 m) und `\unitfrac[1]{m}{s}` (z.B. 1 m/s) gesetzt.

Chapter 3

XBRL

3.1 Overview

TODO

3.2 Concepts

In section **??**, we learned that a fact is the smallest unit of information in an XBRL report. One of the core aspects of a fact is its concept. The concept refers to what is being reported. So for example, if a fact is reporting information about a company's revenue, then the concept of the fact is "Revenue". In this section, we will take a closer look at concepts and how they are defined in XBRL.
Concepts are the fundamental building blocks of XBRL and they are defined in the XBRL taxonomy.

3.2.1 Taxonomy

TODO: the root taxonomy should be called 'extension taxonomy'
The XBRL taxonomy is a collection of concepts. Each XBRL report defines its own taxonomy inside of a taxonomy schema file. This schema file is called the **root schema** of the taxonomy. The root schema is the entry point to the taxonomy and it contains references to all other schemas in the taxonomy. By resolving these references, the entire taxonomy can be loaded into memory.
The schemas being referenced include the following:

- **us-gaap** [0] - Contains concepts for US Generally Accepted Accounting Principles (GAAP).
- **ifrs** [1] - Contains concepts for International Financial Reporting Standards (IFRS).
- **dei** [2] - Contains concepts for the SEC's Document and Entity Information (DEI) requirements.
- **country** [3] - Contains concepts for country codes.
- TODO: add more

[0] https://xbrl.us/us-gaap/
[1] https://www.ifrs.org/
[2] https://www.sec.gov/info/edgar/dei-2019xbrl-taxonomy
[3] https://xbrl.fasb.org/us-gaap/2021/elts/us-gaap-country-2021-01-31.xsd

3.3 Facts

A fact is the smallest unit of information in an XBRL report. The word "Fact" is a term used to describe an individual piiece of financial of business information within an XBRL instance document.
Lets consider a simplified example involving a financial report for a company. Suppose the report contains information about the company's revenue for the fiscal year ending on December 31, 2022.
In XBRL, a corresponding fact would be represented as follows:

- **Concept:** Revenue
- **Entity:** Microsoft Coorporation
- **Period:** from 2022-04-01 to 2023-03-31 [0]
- **Unit:** USD
- **Value:** 198'000'000 [1]

In this example:

- The **concept** refers *what* is being reported. In this case, "Revenue" indicates that the fact is reporting information about the company's revenue.
- The **entity** refers to *who* is reporting. In the case of our example, the entity is "Microsoft Coorporation".
- The **period** refers to *when* the information is being reported. The period is defined as the fiscal year 2022.
- The **unit** refers to *how* the information is being reported. In this example, the unit is "USD", which indicates that the information is being reported in US dollars.
- The **value** refers to *how much* is being reported. According to Microsoft's 2022 annual report, the company's revenue for the fiscal year 2022 was around 198 billion US dollars.

The concept, entity, period and unit of a fact are called its **aspects**. If necessary, additional aspects can be defined for a fact. These additional aspects are called **dimensions** and will be covered in section TODO. The aspects that are not dimensions are called **core aspects**. Even though the name suggests otherwise, core aspects are not all mandatory for a fact. The only mandatory core aspect is the concept.

3.4 QNames

Although the motivation behind this XBRL processor is to shield its user from the complexity of XML, we keep one key aspect of XML in our API: QNames.
QNames are a way to uniquely identify an XML element or attribute. They consist of a local name and a namespace, which in turn consists of a namespace prefix and a namespace URI. The namespace URI is a URI that uniquely identifies the namespace and namespace prefix acts as a shorthand for the namespace.

[0] Refers to the fiscal year 2022, which starts on April 1, 2022 and ends on March 31, 2023
[1] https://www.microsoft.com/investor/reports/ar22/index.html

Figure 3.1: The us-gaap:Assets QName

- Namespace prefix: `us-gaap`
- Namespace URI: `https://xbrl.fasb.org/us-gaap/2022/elts/us-gaap-2022.xsd`
- Local name: `Assets`

For example the QName `us-gaap:Assets` identifies the element `Assets` in the namespace `us-gaap`.

In this example, the namespace prefix `us-gaap` is a shorthand for the namespace URI `https://xbrl.fasb.org/us-gaap/2022/elts/us-gaap-2022.xsd`, and together they form the namespace `us-gaap`.

QNames are used in the XBRL taxonomy to identify concepts, facts and other elements. Since they provide a robust and easy way to identify elements, we decided to use them in our API as well. However, there is one important difference between our QNames and the QNames used in the XBRL taxonomy: In the XBRL taxonomy, the mapping from namespace prefixes to namespace URIs depends on where the QName is used. In our API, there is a fixed, global mapping from namespace prefixes to namespace URIs.

3.4.1 Networks

Up to this point, all concepts in the XBRL taxonomy are treated as independent entities. The whole filing can be viewed as an unordered set of facts with no relation between them. However, in reality, concepts are not independent of each other. Instead, concepts are often related to each other in some way.

For example, the concepts "Assets" and "Liabilities" are related to each other in the sense that they are both part of the concept "Balance Sheet". Furthermore, the concept "Assets" can be further divided into the concepts "Current Assets" and "Non-Current Assets". The afforementioned relations can be visualized using a directed graph, as shown in figure 3.2.

Figure 3.2: Example of a relations between concepts in XBRL

```
Balance Sheet
├── Assets
│   ├── Current Assets
│   └── Non-Current Assets
└── Liabilities
```

Any reader with a basic understanding of computer science will recognize that the above example is a graph, more specifically a tree. Graphs are commonly used to represent relations between entities. In the context of XBRL, graphs are used to represent all kinds of relations between concepts, facts, and other elements of an XBRL filing. From now on, I will refer to graphs that represent relations between concepts as "networks". XBRL commonly refers uses the term "Extended Link" to refer to networks or parts of networks.

3.4.2 Types of Networks

The XBRL 2.1 specification defines 6 built in types of networks[3][1]:

- `link:presentationLink`: A network that represents the hierarchy of concepts in a report. An example of this can be seen in figure 3.2.
- `link:calculationLink`: A network that represents how concepts are calculated from other concepts. For example, in figure 3.2, the concept "Assets" is calculated as the sum of the concepts "Current Assets" and "Non-Current Assets".
- `link:definitionLink`: A network that represents the relations between concepts and their definitions. For example, there might be a concept "TotalAssets" that has the same semantic meaning as the concept "Assets". A definition network would represent that the two concepts are aliases of each other.
- `link:labelLink`: A network that associates report elements with human-readable labels.
- `link:referenceLink`: A network that links report elements to external resources. For example, the concept "Total Shareholder Return Amount" might have an official definition in the SEC's Code of Federal Regulations (CFR). The reference network would link the concept to the subparagraph `17 CFR 229.402(v)(2)(iv)`[7].
- `link:footnoteLink`: A network that links report elements, facts and other elements to footnotes.

XBRL refers to these built in networks as "standard extended links". If needed, XBRL allows users to define their own networks, which are referred to as "custom extended links"[3].
In the subsequent sections, I will describe how networks are implemented in XBRL on a conceptual level.

3.5 Report Elements

In the context of XBRL, report elements are the fundamental building blocks that make up the structure of an XBRL report. Concepts, abstracts, line items, dimensions, members, and hypercubes provide a comprehensive framework for structuring and organizing data in a way that is consistent and machine-readable.

[1]The XBRL 2.1 specification is inconsistent about `link:footNotelink`. Section 1.4 does not list it as a standard extended link, section 3.5.2.4 does. I will assume that it is a standard extended link.

Chapter 4

Implementation

4.1 Overview

TODO

4.2 Implementation of QNames and namespace normalization

4.2.1 Overview

As mentioned in chapter TODO, Brel leverages QNames to identify various elements across the XBRL taxonomy. QNames are a concept that is widely used in XML and XML-based languages, such as XBRL. Thus, for most QNames in Brel, the necessary information can be directly extracted from the corresponding XML elements in the XBRL taxonomy. However, there is an important difference between QNames in XML and QNames in Brel - Namespace bindings.
In XML documents, namespace bindings can be defined on a per-element basis. Child elements inherit the namespace bindings of their parent elements, unless they define their own namespace bindings. This allows for the construction of complex namespace hierarchies, where each element can have its own namespace bindings. In Brel, however, namespace bindings are flat and defined on a global level.
The process of converting this hierarchical structure of namespace bindings into a flat structure is called namespace normalization. Namespace normalization not only flattens the namespace hierarchy, but also resolves collisions in namespace bindings. The motivation for having a flat structure of namespace bindings is that it makes it easier for users to search concepts, types, etc. in an XBRL filing using QNames. Lets say a user wants to search for all facts that are associated with the concept `us-gaap:Assets` in the US GAAP taxonomy.
In Brel, the user can simply search for the QName `us-gaap:Assets` and will find all facts that are associated with this concept. Brel will automatically resolve the prefix `us-gaap` to the corresponding namespace URI. If the report uses the prefix `us-gaap1` instead of `us-gaap`, Brel will still be able to resolve the prefix to the correct namespace URI.
In XML, however, the prefix `us-gaap` might be bound to a different namespace URI in each element. Furthermore, the filer of the XBRL might choose to use the prefix `us-gaap1` in some sections of the XBRL taxonomy. So if the user wants to search for all facts that are associated with the concept `us-gaap:Assets` in the US GAAP taxonomy, they would have to supply a context in which the prefix `us-gaap` is bound to the namespace URI of the US GAAP taxonomy. But in another context, the same

concept might be called `us-gaap1:Assets` instead. This is extremely cumbersome for users and makes it very difficult to search for concepts, types, etc. in an XBRL filing using QNames. It also insufficiantly shields users from the complexity of XML and XML-based languages, such as XBRL.

4.2.2 Namespace hierarchy notation

This section exclusively focuses on the implementation of QNames in Brel and namespace bindings in particular. Since XML documents tend to be very verbose and contain a lot of information that is not relevant for namespace bindings, we will use a custom notation to represent namespace bindings in this section. I will refer to this notation as the *namespace hierarchy notation* and describe it using an example.
The namespace hierarchy notation works as follows:

- Each level of the namespace hierarchy is represented as a single line.
- Depending on the level of the namespace hierarchy, the element name has a different indentation.
- The name of the element is followed by a list of namespace bindings, separated by commas.
- Each namespace binding is represented as a key-value pair, where the key is the prefix of the namespace binding and the value is the namespace URI.
- If an element does not define any namespace bindings, the list of namespace bindings is omitted.
- XML attributes that are not namespace bindings are omitted in this notation.

Take the following XML snippet as an example:

Figure 4.1: Example of an XML snippet where namespace bindings are defined on a per-element basis

```
<element1 xmlns:foo = "http://foo.com" color = "red">
    <element2 xmlns:bar = "http://bar.com"/>
        <element3 color = "blue">
    </element2>
</element1>
<element4 xmlns:baz = "http://baz.com" xmlns:foo = "http://other-foo.co
```

Using the namespace hierarchy notation, we can represent the same namespace hierarchy as follows:

Figure 4.2: Example of the same XML snippet in our custom notation

```
root
├── element1 foo = "http://foo.com"
│   └── element2 bar = "http://bar.com"
│       └── element3
└── element4 baz = "http://baz.com", foo = "http://other-foo.com"
```

As we can see, the namespace hierarchy notation is much more compact than the XML snippet. The `color` attribute of `element1` and `element3` is omitted, since it is not a namespace binding.

Figure 4.4: List of namespace bindings extracted from the flattened namespace hierarchy

```
foo = "http://foo.com"
bar = "http://bar.com"
baz = "http://baz.com"
foo = "http://other-foo.com"
```

4.2.3 Flattening namespace bindings

As mentioned in the previous section, namespace bindings in XML are defined on a per-element basis, which is arranged in a tree structure. In Brel, however, namespace bindings are defined on a global level, which is arranged in a flat structure. Thus, we need to flatten the namespace hierarchy of the XBRL taxonomy into a flat structure.

The process of flattening a tree structure into a flat structure is a common problem in computer science. One of the most common approaches to this problem is to use a depth-first search algorithm. This is also the approach that we use in Brel to flatten the namespace hierarchy of the XBRL taxonomy.

Remember that in XML, child elements inherit the namespace bindings of their parent elements. Thus, when flattening the namespace hierarchy, we need to make sure that all the namespace bindings of a parent are also present in its children, unless the children define their own namespace bindings.

To give an example of flattening, let us flatten the namespace hierarchy from the previous figure TODO.

Figure 4.3: Example of the same XML snippet in our custom notation, flattened

```
root
├─element1 foo = "http://foo.com"
├─element2 foo = "http://foo.com", bar = "http://bar.com"
├─element3 foo = "http://foo.com", bar = "http://bar.com"
└─element4 baz = "http://baz.com", foo = "http://other-foo.com"
```

As we can see, the namespace hierarchy has been flattened into a flat structure, meaning that all elements are on the same level. The order of the elements is determined by the depth-first search algorithm.

Each child element inherits the namespace bindings of its parent element. Thus, the `element2` and `element3` elements inherit the namespace bindings of the `element1` element.

To extract the namespace bindings of this flat structure, we can simply iterate over the elements and extract the namespace bindings of each element. For our example, this would result in the following list of namespace bindings:

4.2.4 Collisions in namespace bindings

An observant reader might have noticed that the list of namespace bindings from the previous section contains two bindings for the `foo` prefix. The first binding is defined as `foo = "http://foo.com"`, whereas the second binding is defined as `foo = "http://other-foo.com"`.

This is called a collision and is not allowed in Brel. The following section describes the different types of collisions and how they are handled in Brel.

4.2.5 Types of collisions

There are three types of collisions that can occur in Brel:

- **Version collision**: Two namespace bindings have the same prefix and the same namespace URI, but different versions of it.

  Example: `foo = "http://foo.com/2022"` and `foo = "http://foo.com/2023"`
- **Prefix collision**: Two namespace bindings have the same prefix, but different *unversioned* namespace URIs.

  Example: `foo = "http://foo.com"` and `foo = "http://other-foo.com"`
- **Namespace URI collision**: Two namespace bindings have the same *unversioned* namespace URI, but different prefixes.

  Example: `foo = "http://foo.com"` and `bar = "http://foo.com"`

Version collision

Version collisions occur when two namespace bindings have the same prefix and the same namespace URI, but different versions of it.
Version collisions are allowed in brel, but they do raise an interesting question: Lets say the creates a QName `foo:bar` to search for a concept in the taxonomy. Also assume that the XBRL filing contains the following namespace bindings:

Figure 4.5: Example of a version collision

```
root
├── element1 foo = "http://foo.com/01-01-2022"
│   └── foo:bar
└── element2 foo = "http://foo.com/01-01-2023"
    └── foo:baz
```

Which namespace URI should be used for the QName `foo:bar`?
The mechanism that Brel implements is straightforward: Use the newest version.
First, Brel will remove all digits, dashes and dots from the URI versions. If the two URI versions are equal after this step, then they are considered the same URI with different versions. In our example, both URI versions transform into `http://foo.com/`.
Second, for each URI version, Brel will extract all numbers and compute their sum. The URI version with the higher sum is considered the newer version. For our example, the sum of the first URI version is 2024, whereas the sum of the second URI version is 2025. Thus, the second URI version is considered the newer version. So if the user searches for the QName `foo:bar`, the URI version `http://foo.com/01-01-2023` will be used.
Even though this mechanism is straightforward, it does have some drawbacks. Namely, theoretically it is easy to trick the mechanism into using an older version of a namespace URI. For example, the URI version `http://foo.com/31-12-2021` would be considered newer than `http://foo.com/01-01-2023`.
However, this is not a problem in practice since version collisions tend to be rare. On top of that, most taxonomies are released on a yearly basis and their URI just contains the year of the release. If the URI only contains the year, then the mechanism works as expected.
Furthermore, the mechanism used for searching and comparing QNames in Brel only uses the prefix and the local name of the QName. Therefore, even if the mechanism would be tricked into using an older version of a namespace URI, it would not affect the search results.

Prefix collision

A prefix collision occurs when two namespace bindings have the same prefix, but different *unversioned* namespace URIs. The following figure shows an example of a prefix collision:

Figure 4.6: Example of a prefix collision

```
root
├─element1 foo = "http://foo.com"
│  └─foo:bar
└─element2 foo = "http://other-foo.com"
   └─foo:baz
```

Prefix collisions are not allowed in Brel. Brel will rename one of the prefixes to avoid the collision. In the case of our example above, Brel will `element2`'s binding to `foo1 = "http://other-foo.com"` and will replace all appropriate QNames with the new prefix.

Figure 4.7: Example of a resolved prefix collision

```
root
├─element1 foo = "http://foo.com"
│  └─foo:bar
└─element2 foo1 = "http://other-foo.com"
   └─foo1:baz
```

If the user searches for a QName `foo:bar`, Brel will both search for `foo:bar` and `foo1:bar`.

Namespace URI collision

A namespace URI collision occurs when two namespace bindings have the same *unversioned* namespace URI, but different prefixes. An example of a namespace URI collision is shown in the following figure:

Figure 4.8: Example of a namespace URI collision

```
root
├─element1 foo = "http://foo.com"
│  └─foo:bar
└─element2 bar = "http://foo.com"
   └─bar:baz
```

Namespace URI collisions are not allowed in Brel. Brel will pick one of the two prefixes as the preferred prefix and will rename the other prefix to avoid the collision. In general, Brel will pick the shorter prefix as the preferred prefix. If both have the same length, Brel will pick the prefix that comes first alphabetically.

In the case of our example, `bar` will be picked as the preferred prefix. Brel will rename the prefix `foo` along with all occurences to `bar`.

Figure 4.9: Example of a resolved namespace URI collision

```
root
├── element1 bar = "http://foo.com"
│   └── bar:bar
└── element2 bar = "http://foo.com"
    └── bar:baz
```

There are some prefixes that are considered special and will always be picked as the preferred prefix, regardless of their length or alphabetical order. These special prefixes do not even have to be defined in the XBRL taxonomy. If there is a namespace binding that points to the same namespace URI as one of the special prefixes, the special prefix will be picked as the preferred prefix.

The following prefixes are considered special:

- `xml = "http://www.w3.org/XML/<year>/namespace"`
- `xlink = "http://www.w3.org/<year>/xlink"`
- `xs = "http://www.w3.org/<year>/XMLSchema"`
- `xsi = "http://www.w3.org/<year>/XMLSchema-instance"`
- `xbrli = "http://www.xbrl.org/<year>/instance"`
- TODO

If, for example, the XBRL filing contains a namespace binding `foo = "http://www.w3.org/2001/XMLSchema` then the prefix `xsi` will be picked as the preferred prefix and all occurences of `foo` will be renamed to `xsi`.

The special prefixes and the corresponding namespace URIs can be configured in in the `nsconfig.json` file.

4.3 Implementation of Facts

4.3.1 Overview

Facts in Brel function very similar to facts in XBRL. However, there are a couple of key differences between XBRL and Brel that are worth highlighting. Before we dive into the details of facts in Brel, let us first look at how facts are represented in XBRL.

4.3.2 Facts in XBRL

In the context of XBRL, facts are represented as XML elements in an XBRL instance document. The following snippet shows an example of a fact in XBRL:

Figure 4.10: Example of a fact in XBRL

```
<us-gaap:ExtinguishmentOfDebtAmount
  contextRef="c-216"
  decimals="-6"
  id="f-727"
  unitRef="usd">
    121000000
</us-gaap:ExtinguishmentOfDebtAmount>
```

Thinking back to how we defined facts in chapter TODO, we can see that the above example does not contain all the information that we defined as being necessary for a fact. In particular, the above example does only contain information about the concept, the value, and the unit of the fact. Both the concept and the unit are merely links to other elements in the XBRL taxonomy, but do not contain any information about the concept or the unit themselves. Furthermore, instead of providing information about the entity and the period, the example only contains a reference to a context element.

4.3.3 Contexts in XBRL

In XBRL, contexts are used to provide information about the entity and the period of a fact. If the fact is part of a hypercube, the context also contains information about the dimensions and members of the fact. Contexts are defined in the instance document using the `context` element.
Going back to our example from above, the context element referenced by the fact is defined as follows:

Figure 4.11: Example of the context referenced by the fact in figure 4.10

```
<context id="c-216">
    <entity>
        <identifier scheme="http://www.sec.gov/CIK">
            0000021344
        </identifier>
        &lt;segment&gt;
            <xbrldi:explicitMember dimension="dei:LegalEntityAxis">
                ko:BottlingOperationsInAfricaMember
            </xbrldi:explicitMember>
        &lt;/segment&gt;
    </entity>
    <period>
        <startDate>2023-01-01</startDate>
        <endDate>2023-06-30</endDate>
    </period>
</context>
```

The XML snippet fills in some of the missing information from the fact, namely:

- The **entity** refers to *who* is reporting. The entity XML element provides us with a identifier for the entity, as well as information about where the identifier comes from. In this case, the identifier is a Central Index Key (CIK) provided by the US Securities and Exchange Commission (SEC). If we look up the identifier "0000021344" in the SEC's database, we can see that it corresponds to the Coca Cola Company.

- The **period** refers to *when* the information is being reported. As described in chapter TODO, a period can either be a point in time or a duration. In this case, the period is defined as a duration, starting on January 1, 2023 and ending on June 30, 2023.
- The **dimensions** refer to additional information about the fact. This information is only relevant if the fact is part of a hypercube. Dimensions are defined as child elements of the `segment` element, which is a child element of the `entity` element.

A observant reader might have noticed that the placement of dimensions in the context element is not consistent with the placement of both the entity and the period information of a fact. The main reason for this inconsistency is that the XBRL specification was not designed with hypercubes in mind. Hypercubes were added to the XBRL specification at a later point in time.
A second oddity of facts in XBRL is that they are not self-contained. Their definition is spread across multiple elements in the instance document. Most noticably, the context element is defined separately from the fact element. Since different XBRL facts reference the same context element, the incentive behind this design decision is clear: Reducing redundancy. A XBRL instance document can contain thousands of facts, but only a handful of context elements.
However, this design falls apart as soon as we consider the case of hypercubes. To illustrate the problem, consider the following numbers:
In their 2023 Q2 report, the Coca Cola Company

1. reported 1658 facts.
2. defined 13 context elements without dimensions.
3. defined 397 context elements with dimensions.
4. used the same entity for all facts.
5. uesd around 15 different periods for all facts.

So the vast majority of all contexts exist to provide information about the dimensions of a fact. However, the entity and the period information are mostly redundant.

4.3.4 Units in XBRL

In the same way that contexts are used to provide information about the entity and the period of a fact, units are used to provide information about the unit of a fact. Units are also defined in the instance document using the `unit` element. Going back to our example from above, the unit element referenced by the fact is defined as follows:

```
<unit id="usd">
    <measure>iso4217:USD</measure>
</unit>
```

3

This XML snippet connects our intuitive understanding of the unit "usd" with a formal definition of the unit. It does so by referencing the official ISO 4217 currency code for the US dollar.

4.3.5 Concepts in XBRL

As mentioned in chapter TODO, concepts are the fundamental building blocks of XBRL. Each fact is associated with exactly one concept. Unlike units, periods, and contexts, concepts are not defined in the instance document. Instead, they are defined in the XBRL taxonomy, which is a collection of XML schema documents. In case of the above example, the concept is defined in the US GAAP taxonomy, which is a collection of XBRL taxonomies for US Generally Accepted Accounting Principles (GAAP). This already gives us a hint about where to look for the definition of the concept.
The following snippet shows the definition of the concept in the US GAAP taxonomy:

```
<xs:element id="us-gaap_ExtinguishmentOfDebtAmount" name="ExtinguishmentOfDebt
```

4

The main purpose of this snippet is to provide a formal definition of the concept and to constrain the values that are allowed for the concept. In particular, it tells us that the concept is a monetary item, that it has a debit balance, and that it has a duration period type.

4.3.6 Facts in Brel

In Brel, facts are represented as Python objects. When processing facts in XBRL, the information about the fact is spread across multiple elements in multiple documents. The main goal of Brel is to provide a more intuitive interface for working with facts. Whereas XBRL facts in XML, concepts, units, periods, etc. are all treated differently, Brel facts treat all aspects of a fact equally.

4.4 Components Implementation

4.4.1 Overview

Chapter 3 introduced the concept of components in XBRL. To recap, components act as chapters of a report and link together facts that belong to the same chapter. In this chapter, we will look at how components are implemented in Brel.[1] Brel closely follows the XBRL definition of components.[1] Throughout this chapter, we will use a running example to illustrate the concepts introduced in this chapter.
The XBRL specification uses the term "role" while Brel uses the term "component" to refer to the same concept.

4.4.2 Components in XBRL

Components in XBRL are defined using the `link:roleType`[2] element. The following snippet shows an example of a component in XBRL:

[3] taken from COCA COLA CO's 2023 Q2 report
[4] taken from the US GAAP taxonomy
[1] At the time of writing, the Open Information Model (OIM) does not define components.
[2] The prefix "link" commonly refers to the linkbase namespace 'http://xbrl.org/2003/linkbase'.

Figure 4.12: Example of the component "Inventory" in XBRL from Tesla Inc.'s 2023 Q3 report[1]

```
<link:roleType id="Inventory" roleURI="http://www.tesla.com/role/Inventory">
  <link:definition>0000010 - Disclosure - Inventory</link:definition>
  <link:usedOn>link:presentationLink</link:usedOn>
  <link:usedOn>link:calculationLink</link:usedOn>
  <link:usedOn>link:definitionLink</link:usedOn>
</link:roleType>
```

I will refer to the component in figure 4.12 as the "Inventory" component. All properties of XBRL components are either required or optional. I will use the notation `(required)` and `(optional)` to indicate the requiredness of a property. The Inventory component has the following properties:

- `roleURI` (required): The URI of the Inventory component. This URI is used to reference the component from other elements in the XBRL taxonomy. It is the primary identifier of the component. The roleURI has to be unique on a per-taxonomy basis.[2]
- `id` (optional): The id of the component, which is commonly used to reference the component from linkbases.[3] [4]
- `definition` (optional): A human-readable description of the component.
- `usedOn`: A list of linkbases that the component is used in. Whenever a linkbase uses a component, it must declare the component in the `usedOn` property. [5] Therefore, the `usedOn` property can be interpreted as a list of possible linkbases that can use the component. However, even though the Invenory component indicates the use of e.g. a presentation linkbase, it does not mean that any presentation linkbase is defined for the component.

Most of the roles in XBRL will not be defined by the creators of the report, but rather by the creators of the XBRL taxonomy. In the case of Tesla Inc.'s 2023 Q3 report, only 60 components are defined by the creators of the report, while roughly 600 components are defined by the creators of the XBRL taxonomy.[1]

4.4.3 Components in Brel

The main difference between components in XBRL and Brel is that Brel abstracts away the `usedOn` property. Instead, components in Brel provide a direct way of accessing the linkbases that use the component. A component in Brel is defined using the `Component` class. The following UML diagram shows the class diagram of the `Component` class:

Figure 4.13: UML diagram of the `Component` class

Note that the `get_info` method refers to the text of the definintion element in the XBRL taxonomy. This renaming was done to avoid confusion with the `get_definition` method that returns the definition network of the component.

[3] Even though the id is optional, it is recommended to provide an id for each component.[2]

[4] Similar to the roleURI, the id has to be unique on a per-taxonomy basis.[8]

[5] The `usedOn` property is only required for standard extended links and standard resource elements.[2]

Even though the `usedOn` property has been discarded, during parsing, Brel will still perform a check for the `usedOn` property. More precisely, if there exists a non-standard network that is associated with the component, Brel will check if the `usedOn` property is defined for the component. If not, Brel will raise an exception.

4.5 Implementation of Networks

4.5.1 Overview

In chapter 3, we introduced the concept of networks. We also looked at the different types of networks that are defined in the XBRL 2.1 specification. In this chapter, we will look at how networks are represented in XBRL and how Brel parses them.
As previously mentioned, one of the main goals of Brel is to shield the user from the complexity of the XBRL specification. Brel achieves this on two different levels.
First, Brel implements a wrapper around each type of network. This wrapper provides a clean interface for some of the most common operations on networks. The functionality of the wrapper depends on the type of network.
For example, the wrapper for calculation networks provides functions to validate the calculation network and to calculate the value of a concept.
Second, Brel provides a simple API that allows users to access and traverse networks independent of the type of network. This API provides methods for traversing any directed acyclic graph. It is intended for advanced users and is useful for debugging networks in a report.
For example, for each network, Brel provides a function that returns all direct children of a node in the network.

Bibliography

[1] Tesla Inc. 10-q quarterly report for quarter ending september 30, 2023. `https://www.sec.gov/ix?doc=/Archives/edgar/data/1318605/000162828023034847/tsla-20230930.htm`, 2023.

[2] XBRL International Inc. Extensible business reporting language (xbrl) 2.1. `http://www.xbrl.org/specification/xbrl-2.1/rec-2003-12-31/xbrl-2.1-rec-2003-12-31+corrected-errata-2013-02-20.html#_5.1.3`, 2013.

[3] XBRL International Inc. Extensible business reporting language (xbrl) 2.1 terminology. `http://www.xbrl.org/specification/xbrl-2.1/rec-2003-12-31/xbrl-2.1-rec-2003-12-31+corrected-errata-2013-02-20.html#_1.4`, 2013.

[4] Miomir irgendwer ć and Branislav Borovac. Zero-moment point — thirty five years of its life. *International Journal of Humanoid Robotics*, 1(01):157–173, 2004.

[5] Gill A. Pratt and Matthew M. Williamson. Series elastic actuators. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3137–3181, 1995.

[6] Marc Raibert. *Legged Robots That Balance*. MIT Press, Cambridge, MA, 1986.

[7] U.S. Securities and Exchange Commission (SEC). 17 cfr § 229.402 - (item 402) executive compensation. `https://www.ecfr.gov/current/title-17/part-229#p-229.402(v)(2)(iv)`, 2023.

[8] World Wide Web Consortium (W3C). xml:id version 1.0. `https://www.w3.org/TR/xml-id/`, 2005.

Appendix A

Irgendwas

Bla bla ...