ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Autonomous Systems Lab
Prof. Roland Siegwart

Master's Thesis

Brel

Autumn Term 2024

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

This thesis was written as part of my master's degree in computer science at ETH Zurich. It was supervised by Prof. Gustavo Alonso and Dr. Ghislain Fourny.
Both the thesis and the source code of the library are available on GitHub[2]
The contents of this thesis are based heavily on both the XBRL standard[6] created by Charles Hoffman and "The XBRL Book"[5] by Ghislain Fourny.

[2] `https://github.com/PapediPoo/Brel`

Abstract

Hier kommt der Abstact hin ...

Chapter 1

Introduction

1.1 Introduction

In the era of data-driven decision making, the ability to efficiently interpret and analyze business reports is becoming increasingly important. Until 20 years ago, most business reports were published in paper form. This made it difficult for computers to automatically process the information contained in the reports. However, this changed with the introduction of the eXtensible Business Reporting Language (XBRL) in 2003. XBRL is a standardized format for business reports that is both human-readable and machine-readable. Originally designed for financial reports, XBRL is now used in a wide variety of business reports. Initially, XBRL was based on XML, but it has since been extended to support other formats such as JSON and CSV.

Back in the day, XBRL was a niche technology that was only used by a handful of companies. Now, XBRL is gaining traction in both the public and private sectors. Reporting authorities such as the US Securities and Exchange Commission (SEC) and the European Banking Authority (EBA)[2] are increasingly requiring companies to submit their reports in XBRL format.[16] In the private sector, XBRL is used by companies such as JP Morgan Chase, Microsoft, and Hitachi to automate their financial reporting processes.[3]

However, XBRL is not without its problems. As a standard, XBRL is very complex and difficult to understand. Also, many of its intricacies are a result of legacy design decisions and make working with XBRL unnecessarily difficult. On top of that XBRLs design is closely tied to the XML format.

Since XBRL is a tool primarily designed for non-technical users, it is important that it is easy to use. However, the current state of XBRL is far from ideal. There are lots of different XBRL tools, most of which are proprietary and closed-source. Some open source tools exist, but apart from the XBRL platform Arelle[15], they are either incomplete or poorly documented.

In 2021, XBRL International published a partial rewrite of the XBRL standard called Open Information Model (OIM)[14]. The goal of OIM is to learn from the mistakes of the past and to make XBRL more accessible to both developers and non-developers. Compared to the original XBRL standard, the OIM is not strictly tied to the XML format. The OIM is a major step in the right direction, but it is still in its infancy and only reworks the core XBRL concepts. Also, at the time of writing, there are no open source implementations of the OIM, which is where this thesis comes in.

1.2 Goals of this thesis

The goal of this thesis is to create a new open source XBRL library that is based on the OIM. The name of the library is `Brel` (short for **B**usiness **R**eporting **E**xtensible **L**ibrary) and it is written in the Python programming language. The library should be easy to use and well documented. It should provide a simple python API that allows developers to easily read XBRL reports and extract information from them. The library should also provide basic functionality for validating XBRL reports. Finally, the library should support XBRL reports in XML, but should also be able to support other formats such as JSON and CSV in the future.
The research questions that this thesis aims to answer are:

- How can the OIM be translated into an easy-to-use python API?
- How should the non-OIM sections of XBRL be converted into a python API that is consistent with the OIM?
- How can the library be designed to support multiple formats in the future?

1.3 Structure of this thesis

Since this thesis is about XBRL, it is important to first understand what XBRL is and how it works. Therefore, chapter 3 will give a brief introduction to XBRL. It will introduce the core concepts of XBRL in the OIM and move on to the non-OIM sections of XBRL. It will not dive deep into the technical details of the XBRL standard, but instead focus on the concepts that are relevant for this thesis.
Next, chapter 4 will describe both the design and the implementation of the library. It will explain how the OIM is translated into a python API and how the non-OIM sections of XBRL are converted into a python API that is consistent with the OIM. Furthermore, it will cover the complexities of translating XBRL reports in XML format into python objects. Finally, it will explain how the library is designed to support multiple formats in the future.
In the results chapter **??**, the library will be evaluated based on the goals set out in the introduction. This chapter will also give a comprehensive overview of the library's features and limitations. Additionally, it will give examples of how the library can be used to read and validate XBRL reports.
Lastly, chapter **??** will summarize the results of this thesis and give an outlook on future work.

Chapter 2

Related work

2.1 XBRL

As mentioned in section 1, this thesis is based on the XBRL standard[6] originally created by Charles Hoffman. The XBRL standard is a complex standard consisting of many different parts. The parts of the XBRL standard that are relevant to this thesis are the Open Information Model (OIM)[14], the XBRL 2.1 specification[8], the extension for dimensional reporting[9] and the specification for generic links[10]. The XBRL 2.1 specification and the OIM overlap in some areas, but the OIM is a partial rewrite of the XBRL 2.1 specification. It does not contain all the features of the XBRL 2.1 specification, but it is a lot easier to understand. The OIM also covers some aspects of XBRL dimensions, but does not contain any features from XBRL generic links.

2.2 The XBRL Book

To properly understand the XBRL specification, one already needs to have a good understanding of both XML and XBRL. This makes it difficult for beginners to get started with XBRL. To address this problem, Dr. Ghislain Fourny wrote "The XBRL Book"[5]. The book is a comprehensive guide to XBRL and covers all the important aspects of the XBRL standard, including the rather recent OIM.

2.3 Arelle

Arelle[15] is an open source XBRL platform written in Python. At the time of writing, Arelle is the most complete open source XBRL platform. It supports all the features of the XBRL 2.1 specification and the OIM. Like Brel, Arelle is also written in Python and is open source. Unlike Brel, Arelle is a complete XBRL platform and not just a python library.

2.4 Xule

Xule[18] is a rule language for XBRL. It is a declarative language that allows users to write rules that can be used to validate XBRL reports. Xule is written in Python and is open source. It is also part of the Arelle project and is used by Arelle to validate XBRL reports. Think of Xule as a domain-specific language for XBRL validation rules. Even though Xule is not part of this thesis, Brel should be able to support Xule in the future.

Chapter 3

XBRL

3.1 Overview

In essence, XBRL is a standardized format for representing reports. XBRL is an acronym for eXtensible Business Reporting Language.
As Ghislain Fourny has put it in *the XBRL Book* [5]:

> If XBRL could be summarized in one single definition, it would be this:
> XBRL is about reporting facts.

Keeping this in mind, this chapter will first introduce the basic concepts of XBRL, namely facts, concepts and QNames. Then, it will introduce the more advanced concepts that are about putting facts into relation with each other, namely roles, networks, and report elements.
Armed with this fundamental knowledge about XBRL, you will then be able to understand how Brel implements the core parts of the XBRL standard and how it hides a lot of the complexity of XBRL behind a simple Python API.
This chapter will not cover the XBRL specification in its entirety. It will also gloss over a lot of the details of the XBRL specification. It is more focused on giving the reader a high-level overview of the XBRL specification.
Furthermore, a lot of concepts in XBRL require knowledge of other XBRL concepts. There are also a few circular dependencies between the concepts, which makes it hard to explain them in a linear fashion. Therefore, there are a few sections in this chapter that will redefine concepts that have already been introduced. This is done to gradually introduce the reader to the more complex concepts of XBRL.

3.2 Facts

A fact is the smallest unit of information in an XBRL report. The word "Fact" is a term used to describe an individual piece of financial of business information within an XBRL instance document.
Lets consider a simplified example involving a financial report of Microsoft Coorporation for the fiscal year 2022. Microsoft's annual report can be found on the company's website[0]. It contains a lot of information about the company's financial situation, as well as information about the company's business activities. For this example, we will only consider the company's revenue for the fiscal year 2022. Microsoft chose to report this information as follows:

[0] https://www.microsoft.com/investor/reports/ar22/index.html

Microsoft | Annual Report 2022

SUMMARY RESULTS OF OPERATIONS

(In millions, except percentages and per share amounts)	2022	2021
Revenue	$ 198,270	$ 168,088
Gross margin	135,620	115,856
Operating income	83,383	69,916
Net income	72,738	61,271
Diluted earnings per share	9.65	8.05
Adjusted net income (non-GAAP)	69,447	60,651
Adjusted diluted earnings per share (non-GAAP)	9.21	7.97

Figure 3.1: Microsoft's summary results of operations for the fiscal year 2021 and 2022

[4]

This table contains multiple facts about Microsoft for the fiscal years 20201 and 2022 as seen by the horizontal axis. The vertical axis describes what is being reported. The "what is being reported" is called the **concept** of a fact. It reports the values for the concepts "Revenue", "Cost of revenue", ... for both fiscal years. In summary, the table contains 14 facts across 7 concepts for 2 fiscal years.
For now, let us focus on the top left fact, which reports the company's revenue for the fiscal year 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:** 198'270'000 [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". In our example this is implicit, since we are looking at Microsoft's annual report. However, this has to be explicitly stated in an XBRL report.
- The **period** refers to *when* the information is being reported. The period is defined as the fiscal year 2022. Again, this information is implicit in our example.
- 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. This is indicated by the dollar sign $ in the table.
- 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.

[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

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.

3.3 Concepts

In section 3.2, 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 what is called the **taxonomy**.

3.3.1 Taxonomy

Simply put, the XBRL taxonomy is a collection of concepts. Each XBRL report defines its own taxonomy inside of a taxonomy schema file. The taxonomy defined by the report is called the **extension taxonomy**.
This extension taxonomy contains references to other taxonomies, which may contain references to even more taxonomies. So when a report and its extension taxonomy are loaded into memory, the entire taxonomy is loaded into memory. The transitive closure of all these references is called the **DTS** (short for **D**iscoverable **T**axonomy **S**et).
Note that most of the taxonomies in the DTS are not located on the same machine as the report. Instead, they are located on the internet and are downloaded on demand.
Some taxonomies commonly found in a report's DTS are:

- **us-gaap** [1] - Contains concepts for US Generally Accepted Accounting Principles (GAAP).
- **ifrs** [2] - Contains concepts for International Financial Reporting Standards (IFRS).
- **dei** [3] - Contains concepts for the SEC's Document and Entity Information (DEI) requirements.
- **country** [4] - Contains concepts for country codes.
- **iso4217** [5] - Contains concepts for currency codes.

Since a lot of DTSs from different reports share a lot of the same taxonomies, it makes sense to cache the taxonomies locally.

[1] https://xbrl.us/us-gaap/
[2] https://www.ifrs.org/
[3] https://www.sec.gov/info/edgar/dei-2019xbrl-taxonomy
[4] https://xbrl.fasb.org/us-gaap/2021/elts/us-gaap-country-2021-01-31.xsd
[5] https://www.iso.org/iso-4217-currency-codes.html

3.3.2 Concepts

Each concept in the DTS is identified by a **QName**. The technical intrecacies of QNames will be covered in section 3.4, but for now think of them as a unique identifier for a concept. The QName of a concept tends to be human-readable and self-explanatory. However, accountants and business analysts tend to go overboard with with their naming conventions. Some examples of the QNames of concepts are:

- `us-gaap:Assets`
- `ko:IncrementalTaxAndInterestLiability`
- `dei:EntityCommonStockSharesOutstanding`
- `us-gaap:ElementNameAndStandardLabelInMaturityNumericLowerEndToNumericHigherEndDateMea`

But concepts do not only consist of a QName. They also constrain some of the aspects and values of the facts that reference them. Going back to our running example from section 3.2, the concept `us-gaap:Revenue` introduces some constraints. For example, it constrains the value to be a `monetaryItemType`. This means that the value of the fact must be a number and not just any arbitrary string.[6] It also constrains the unit to be a currency defined in the ISO 4217 standard.[1] Moreover, monetary facts have to be labeled as either "debit" or "credit" using the `balance` aspect. In this case, the concept constrains the fact to have a "debit" balance, since the company's revenue is an asset.
The concept `us-gaap:Revenue` also constrains the period of the fact to be of type "duration". This means that the period of the fact has to be a duration of time, such as a fiscal year or a quarter. Alternatively, the period could be of type "instant", which means that the period refers to a specific point in time.
Finally, the concept `us-gaap:Revenue` allows the fact to be null, which means that there does not have to be a fact for the concept `us-gaap:Revenue`. The vast majority of concepts allow facts to be null.
This wraps up our discussion about concepts in XBRL. In the next section, we will take a look at QNames and how they are used in XBRL. Together with the knowledge about concepts and facts, this will give us a solid foundation to understand the core parts of the XBRL standard.

3.4 QNames

Although the motivation behind the XBRL processor Brel 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 three things: a namespace prefix, a namespace URI, and a local name. The prefix acts as a shorthand for the namespace URI.
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`.

[6] The `monetaryItemType` has additional constraints besides being an integer, but we will ignore them for now.

Figure 3.2: 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`

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: Currently, most XBRL filings are based on XML, where namespace prefixes are defined on a per-element basis. Therefore, 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. The motivation behind this decision is that it makes the API easier to use. The how and why of this mapping will be explained in section 4.2.

3.5 Segway to advanced XBRL

This concludes our overview of QNames as well as the overview of the core concepts of XBRL. Armed with this knowledge, we could already create funcional XBRL reports, albeit with a lot of limitations.

The most glaring limitation is that the facts in the report are not structured in any way. They are just a bunch of facts that are not related to each other. Since facts are not related to each other, it is impossible to verify if the values within the report are consistent.

Besides that, XBRL in its current form is not very user-friendly. The concepts for example are named using QNames, which are not very user-friendly to read. Another limitation is that they are mostly in English, which makes it hard for non-English speakers to understand the report.

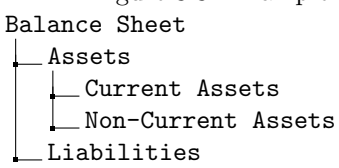
All of these issues will be addressed in the next sections, which will introduce the advanced concepts of XBRL, the most important of which are networks.

3.5.1 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 3.3.

Figure 3.3: Example of a relations between concepts in XBRL



A reader with a basic understanding of mathematics will recognize that the above example is a directed acyclic graph (DAG). More specifically, the above example is 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.

3.5.2 Types of Networks

The XBRL 2.1 specification defines 6 built in types of networks[13][1]:

- `link:presentationLink`: A network that represents the hierarchy of concepts in a report. An example of this can be seen in figure 3.3.
- `link:calculationLink`: A network that represents how concepts are calculated from other concepts. For example, in figure 3.3, 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)`[17].
- `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`[13].
Technically speaking, XBRL does allow networks in XBRL to contain both directed and undirected cycles. However, in practice, networks in XBRL are almost always directed acyclic graphs (DAGs).
In the subsequent sections, I will describe how networks are implemented in XBRL on a conceptual level.

[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.

3.5.3 presentationLink

The `link:presentationLink` network is used to represent the hierarchy of concepts in a report. I will describe presentationLinks in more detail compared to the other networks, since the other network types are implemented in a similar way.
XBRL implements all its networks as a set of directed edges called `arcs`. Each arc has a source and a target. Duplicate arcs are not allowed.
Taking the example from figure 3.3, the presentationLink network would be represented as the following edge list:

Figure 3.4: Example of a presentationLink network in edge list format

```
Balance Sheet -> Assets
Assets -> Current Assets
Assets -> Non-Current Assets
Balance Sheet -> Liabilities
```

Each arc in the example 3.4 is represented as a `link:presentationArc` element in XBRL. Additionally, presentationLinks contain so called "locators" `link:loc` that represent the nodes in the network. In case of a presentaton network, the locators are references to the concepts in the XBRL taxonomy. In other networks, locators can be references to other elements, such as facts.
In case of the XML syntax, the first `link:presentationArc` "Balance Sheet -¿ Assets" would be represented as follows:

```
<link:loc
     xlink:type="locator"
     xlink:href="file_1.xsd#BalanceSheet"
     xlink:label="BalanceSheet_loc"
/>
<link:loc
     xlink:type="locator"
     xlink:href="file_1.xsd#Assets"
     xlink:label="Assets_loc"
/>
<link:presentationArc
     xlink:type="arc"
     xlink:arcrole="http://www.xbrl.org/2003/arcrole/parent-child"
     xlink:from="BalanceSheet_loc"
     xlink:to="Assets_loc"
     order="1"
/>
```

Figure 3.5: Example of a presentationArc in XML syntax

The `xlink:from` attribute of the `link:presentationArc` element references the `xlink:label` attribute of the `link:loc` element that represents the source of the arc. The same applies to the `xlink:to` attribute of the `link:presentationArc` element.
Both locators contain a `xlink:href` attribute that references the concept in the XBRL taxonomy that the locator represents. This reference can even point to a

different file, as shown in figure 3.5.
The `xlink:type` on the elements helps the XML parser to determine the type of the element. Note that the XML tag `link:presentationArc` already hints at the fact that this element is a type of arc. However, the `xlink:type` attribute is still required and provides parsers with a more reliable way to determine if an element is an arc, a locator, or something else.
The `order` attribute of the `link:presentationArc` element specifies in which order the children of the source of the arc should be displayed.
Finally, the `xlink:arcrole` attribute of the `link:presentationArc` element specifies the kind of relation between the source and the target of the arc. In case of a presentationLink, the `xlink:arcrole` attribute is always set to `parent-child`. For different networks, the `xlink:arcrole` attribute can be set to different values. It can even be set to different values for different arcs in the same network.
XBRL packages the set of arcs and locators into a `link:presentationLink` element. This element contains a `xlink:role` attribute that specifies the role of the presentationLink. Roles are a more advanced concept that I will introduce in section 3.6. For now, think of the `link:presentationLink` element as a container for the arcs and locators.

3.5.4 Motivation of Report Elements

Up to this point, I have only described how presentation networks are implemented in XBRL. I also mentioned that presentation networks indroduce a hierarchy of concepts, but this is not entirely true.
I have introduced concepts as the "what"-part of a fact. For example, if the company Foo reports a revenue of 1000 USD in 2019, the concept `Revenue` is the "what"-part of the fact.
However, if we look at the presentation network in figure 3.3, not all of the elements in the network can have a fact associated with them. For example, the concept `BalanceSheet` cannot have a fact associated with it. In XBRL, concepts that can not have a fact associated with them are called `Abstract`.
The Open Information Model (OIM) combines abstracts and concepts using the term `report element`. Report elements are, as the name suggests, elements that can appear in a report. Some of these report elements are used for facts, namely the concepts. Others are used to represent the structure of the report, namely the abstracts. There are six types of report elements in total[14]. I will introduce them as they come up in the subsequent sections.
With the introduction of report elements, our notion of a presentation network changes slightly. Instead of introducing a hierarchy of concepts, presentation networks introduce a hierarchy of report elements. However, our notion of a fact stays the same. A fact is still requires a concept, not a report element.

3.5.5 calculationLink

The `link:calculationLink` network is used to represent how concepts are calculated from other concepts. More specifically, it is used to represent how a concept is the sum of other concepts. Under the hood, calculationLinks are implemented in the same way as presentationLinks, but there are a few differences in the semantics.

1. The arcs now have the tag `link:calculationArc` instead of `link:presentationArc`.
2. The link now has the tag `link:calculationLink` instead of `link:presentationLink`.
3. The `xlink:arcrole` attribute of the `link:calculationArc` element is set to `summation-item`.

4. The `link:calculationArc` element has an additional attribute called `weight`.

5. All locators in the link are references to concepts, not just report elements.

Most of these differences are self-explanatory and do not have any semantic implications. However, the last two differences are worth explaining in more detail.
The main motivation behind calculation networks is so that XBRL processors can either calculate the value of a concept or check if the value of a concept is computed correctly and consistently. In the case of our XBRL processor Brel, the main focus is on the latter. Chapter 6.4 of "The XBRL Book" [5] describes the different consistency checks in more detail.
Concepts within a calculation network are computed as a weighted sum of their children. The `weight` attribute of the `link:calculationArc` element specifies the weight of the child in the sum. Additionally, facts can only be associated with concepts, not just any report element. Therefore, all locators in a calculation network are references to concepts.
In the section on concepts3.3, I have introduced the `balance` aspect of a concept. It specifies if the concept is a debit or a credit. The XBRL 2.1 specification enforces some constraints on the `balance` aspect of concepts in combination with the `weight` attribute of the `link:calculationArc` element [11]. If one concept has a `balance` of `debit` and another concept has a `balance` of `credit`, then their connecting arc must have a negative `weight`. If both concepts have the same `balance`, then their connecting arc must have a positive `weight`.

Concept 1	**Concept 2**	**Connecting edge weight**
Debit	Credit	≤ 0
Credit	Debit	≤ 0
Debit	Debit	≥ 0
Credit	Credit	≥ 0

Figure 3.6: Balance and weight constraints in calculation networks

A network that is consistent with the balance and weight constraints is called a `balance consistent network`.[5]
Balance consistency is not the only kind of consistency that calculation networks can be checked for. Another kind of consistency is `roll-up consistency` which comes in two flavors: `simple roll-up consistency` and `general roll-up consistency`. `Simple roll-up consistency` is a weaker form of roll-up consistency. It requires a calculation network as well as a presentation network and checks if the structure of the two networks is consistent.
TODO

3.6 labelLink

The `link:labelLink` network is used to associate report elements with human-readable labels. Thus far,

3.7 Roles

3.8 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, dimen-

sions, members, and hypercubes provide a comprehensive framework for structuring and organizing data in a way that is consistent and machine-readable.

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"`

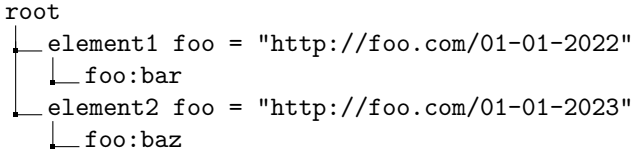
- **Namespace URI collision**: Two namespace bindings have the same *unversioned* namespace URI, but different prefixes.

  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

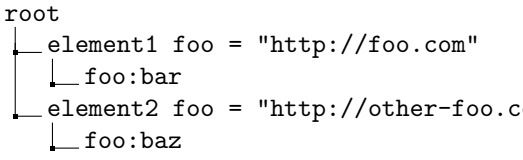


Which namespace URI should be used for the QName `foo:bar`?
The mechanism that Brel implements is straightforward: Use the newest version.
First, Brel will remove all digits, dashes and dots from the URI versions. If the two URI versions are equal after this step, then they are considered the same URI 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

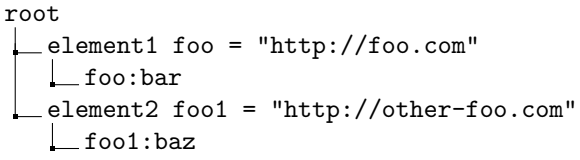
A prefix collision occurs when two namespace bindings have the same prefix, but different *unversioned* namespace URIs. The following figure shows an example of a prefix collision:

Figure 4.6: Example of a prefix collision



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

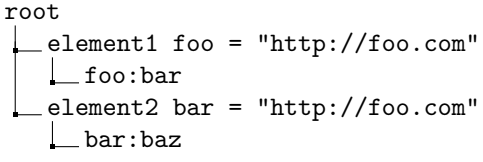


If the user searches for a QName `foo:bar`, Brel will both search for `foo:bar` and `foo1:bar`.

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



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>
```

1

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>
        <segment>
            <xbrldi:explicitMember dimension="dei:LegalEntityAxis">
                ko:BottlingOperationsInAfricaMember
            </xbrldi:explicitMember>
        </segment>
    </entity>
    <period>
        <startDate>2023-01-01</startDate>
        <endDate>2023-06-30</endDate>
    </period>
</context>
```

2

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.[7] 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[7]

```
<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.[12]
- `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.[7]

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.[12]
[4] Similar to the roleURI, the id has to be unique on a per-taxonomy basis.[19]
[5] The `usedOn` property is only required for standard extended links and standard resource elements.[12]

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] European Banking Authority. Eba xbrl filing rules. https://extranet.eba.europa.eu/sites/default/documents/files/documents/10180/2185906/63580b57-b195-4187-b041-5d0f3af4e342/EBA%20Filing%20Rules%20v4.3.pdf?retry=1, 2018.

[2] European Banking Authority. Reporting frameworks. https://www.eba.europa.eu/risk-and-data-analysis/reporting-frameworks, unknown.

[3] Richard Smith Bruno Tesnière and Mike Willis. *the journal*. price waterhouse coopers, 2002.

[4] Microsoft Corporation. Annual report 2022. https://www.microsoft.com/investor/reports/ar22/index.html, 2022.

[5] Dr. Ghislain Fourny. *The XBRL Book: Simple, Precise, Technical*. Independently published, 2023.

[6] Charles Hoffman. Extensible business reporting language (xbrl). https://www.xbrl.org/, 2003.

[7] 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.

[8] XBRL International Inc. Extensible business reporting language (xbrl) 2.1. https://www.xbrl.org/Specification/XBRL-2.1/REC-2003-12-31/XBRL-2.1-REC-2003-12-31+corrected-errata-2013-02-20.html, 2003.

[9] XBRL International Inc. Extensible business reporting language (xbrl) dimensions 1.0. https://www.xbrl.org/specification/dimensions/rec-2012-01-25/dimensions-rec-2006-09-18+corrected-errata-2012-01-25-clean.html, 2006.

[10] XBRL International Inc. Extensible business reporting language (xbrl) generic links 1.0. https://www.xbrl.org/specification/gnl/rec-2009-06-22/gnl-rec-2009-06-22.html, 2009.

[11] XBRL International Inc. Extensible business reporting language (xbrl) 2.1 - 5.1.1 concept definitions. 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#_5.1.1, 2013.

[12] XBRL International Inc. Extensible business reporting language (xbrl) 2.1 - 5.1.3 defining custom role types - the roletype element. 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.

[13] XBRL International Inc. Extensible business reporting language (xbrl) 2.1 terminology - 1.4 terminology (non-normative except where otherwise noted). 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.

[14] XBRL International Inc. Open information model 1.0. https://www.xbrl.org/Specification/oim/REC-2021-10-13+errata-2023-04-19/oim-REC-2021-10-13+corrected-errata-2023-04-19.html#term-component, 2021.

[15] Mark V Systems Limited. Arelle - open source xbrl platform. https://arelle.org/arelle/,, 2010.

[16] U.S. Securities and Exchange Commission (SEC). Inline xbrl. https://www.sec.gov/structureddata/osd-inline-xbrl.html, 2018.

[17] 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.

[18] XBRL US. Xbrl us xule. https://xbrl.us/xule/, 2021.

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

Appendix A

Irgendwas

Bla bla ...