xBRL Quickguide

understanding the xml behind it.

Appie Verschoor

Contents

1 Introduction

eXtensible Business Reporting Language or XBRL for short. I do not pretend to give you a complete write-up on XBRL and Discoverable Taxonomy Sets (DTS). The details are hairy, the implementations are pluriform, and the perception is that it is difficult—and from an implementer's point of view this view is to a great extent correct. To know exactly what you can and cannot do when modelling your taxonomy you must rely on the technical specifications. These are as the name implies technical and broad: the core XBRL 2.1 specification, plus separate modules for calculations, dimensions, tables, formulas, and more. To read them you got to have a clear understanding of XML.

That does **not** mean you cannot acquire any knowledge of XBRL. The basis is pretty simple, but it comes with its own terminology, so do not be put off by terms such as *taxonomy*, *dimension*, or *link-role*. They are technical names for concepts you already understand—*data model* or the *axis* of a two- or three-dimensional diagram. Along the way I will use the XBRL terminology and explain what it means.

I am a software developer by profession and in the past years I had to deal with a lot of xbrl, xml and taxonomies. The first thing I noticed, besides it involves a lot of accounting, how little was written about XBRL from a technical point of view. You have the specifications and almost no literature on how to create a taxonomy by hand. With this guide I hope to make the creation of an instance, and how an instance relates to a taxonomy, especially the technical part, more understandable.

In the first part of this quick guide I will give you the *need-to-knows* when dealing with XBRL reports, or *instances*, as we also call them. That part will contain as little XML as possible, for many reasons, but most because XML is only the tool, and other formats like JSON are on their way. What matters is how a report or filing is built and how the taxonomy holds (almost) all the information needed to fill and file the report.

In the second part, starting with [Taxonomy] I will dive deeper. I will explain the XML and show how to construct different parts of a taxonomy—again with the minimum amount of XML, because software tooling can do the hard work for you.

At the end you will find a [Glossary] and some [Useful books, websites and tools].

2 Facts

> *In the end, we are only interested in the facts.*

XBRL serves a special purpose: it automated the process and made filed reports machine-readable and -interpretable. A couple of decades ago most filings were on paper—non-indexable and non-searchable—so only a handful of people ever read them. The next step was the same report as PDF or HTML, searchable, but the information inside was still not indexable. XBRL standardized the final step and made filings and reports searchable, indexable and comparable. Nowadays it is easy to compare the net revenue of different companies, the amount of shares held by companies, or any other fact that can be stated on a report. All data structured to the same model, available in machine readable format.

Many organisations are required to publish one or more reports each year, quarter, half-year, by different authorities, like the government, federal tax, oversight committees, continental entities and so on. All the reports or filings have the same thing in common, they state facts. Facts based on the prescribed taxonomy or "data model". What makes, in XBRL terms, a fact?

Let us look at a possible fact from *Cooljapan* a small toy company.

`Net revenue (€ millions), 2024, 1079.5`

The *value* is `1079.5`, we see that's reported in `millions of euro`, we call that a *unit* in XBRL, the value was valid for the *period* `2024` and the *entity* reporting this fact is `Cooljapan`.

To put it differently, a common fact consists of

```
Value: 1079.5
What: Net revenue
Who: Cooljapan
When: 2024
of what: Millions of euro.
```

And to put it in xBRL terms:

```
Value: 1079.5
Concept: Net revenue
Entity: Cooljapan
Period: 2024
unit: Euro
decimals: -6
```

We see that the *what* is called *[Concepts]*, the *who* is referred to as *entity*, the when as *period* and the *of what* has a formal name *unit* and the *decimals=-6* tells that we report in millions of the *unit*. In XBRL we call these different properties of a fact *aspects*.

The combination of the value and the different aspects is what makes a fact. And in xBRL all that we report are facts. Which values can and must be reported is defined by the [taxonomy]. We will look at this in great detail in part II the technical nitty gritty.

A report, or instance, will contain more than just one fact, there will be many. One way to present the facts is in a two-dimensional table called the *fact table*. In a fact table each fact is represented by a row. Each aspect is a column. A very simple filing with only three facts stated could have a fact table like the example below.

concept	value	entity	unit	period
net revenue	200	Cooljapan	euro	2024
gross sales	300	Cooljapan	euro	2024
investments	150	Cooljapan	euro	2024

a fact table with three different concepts

The fact table is just one way to look at a report. Later we will see that there are different types of tables that can help us to make an XBRL instance human readable and group related facts together. These aides however are not needed to create a report.

We saw an example of what is called a monetary fact, but we can report on much more than only monetary facts. We can report text, areas of land, number of units in an appartement block, etc. All what matters is that the facts we report on are found in the taxonomy that is used to create the report and that all the required aspects are provided.

In the next parts we'll dive deeper into the different aspects that make a fact, starting with the Concept.

3 Concepts

As we saw in the previous chapter a concept is an aspect of a fact and it is *the what* that is reported. *Gross Sales*, *Net Revenue*, "*Average number of employees*" are all examples of concepts. Concepts come, by the specification, in two flavors, There are Concepts and Abstract Concepts, which I will also refer to as just *Abstracts*. The difference between 'real concepts' and abstracts is that you can only report values on real concepts.

Abstracts on the other hand are only used to add structure to a report. They can be used for titles and headings in a report. We will get to know them when we properly introduce them in the second part when we add structure to the taxonomy. For now I will focus on the real concepts. Concepts represent the building blocks of a report and they are the only obligatory parts of a report. You cannot create an instance without a concept, while on the other hand in theory all other aspects can be absent from the report.

Concepts are defined in a taxonomy which usually gets published by an authority like the federal tax agency or the house of commerce. These concepts can be found in a schema file and are defined as XML Elements. Each concept must have an attribute that tells us if the concept is an **abstract** or not. It also must have both a **name** and an **id**. There also needs to be an attribute that tells us what type of **period** needs to be reported; *period* or *duration*, what **type** of *unit* is used for the concept, which **substitutiongroup** can be used. The **nillable** attribute tells us if it is allowed to report on the concept without providing a value. If a concept is of a Monetary type you must also provide the **balance** of the value, either debit/credit.

```
<xs:element abstract="false"
    id="jenv-bw2-i_Assets"
    name="Assets"
    nillable="false"
    substitutionGroup="xbrli:item"
    type="nl-types:monetaryNoDecimals20ItemType"
    xbrli:balance="debit"
    xbrli:periodType="instant" />
```

Example of a Concept definition from the Dutch taxonomy (NT)

Let's analyse the definition of the 'Assets' concept.

- We see that it's a reportable item because the *abstract* attribute is false.
- By the name of the type we can deduct that it's a monetary unit which does not allow for decimals (so only integers) the definition of this *nl-types:monetaryNoDecimals20ItemType* can be found elsewhere in the taxonomy so machines also know what we've deducted from the name.
- If you report on this fact, you must provide a value because **nillable** is false.
- The value entered is by definition *debit*.
- And for the period type you must provide a single date because the definition is *instant*.
- The *substitutionGroup* in this case is `xbrli:item`, you can ignore this attribute for now. Reportable items are of the substitutionGroup `xbrli:item`. Be aware that there are other substitutionGroups and they confine the subset of items one can choose to use for a value.

Each concept that is found in the taxonomy has a definition comparable to the one shown. There are of course subtle differences which are expressed by the attributes. Now it is possible to add information to this concept. This can be done by Labels, Documentation and references. These elements are of course important, but they do not add to the report. They exist to make the taxonomy readable and meaningful. I'll skip these for the moment and come back to them when we introduce the taxonomy.

In brief so far:

- Reports consist of Facts,
- Facts consist of aspects and a value,
- Concepts are the main aspect of a Fact.
- The other aspects that are almost always present are
  - Entity (who)
  - Period (when)
  - Unit (of what)

4 Namespaces

A more technical intermezzo that is needed, namely *namespaces*. Now you all heard of this term before, because it's used in software development, deployments and plenty of other places. Namespaces are used to confine the use of a name to a certain domain. This enables you to use different domains, which potentially have clashing names inside them. By confining them to a namespace it is possible to distinguish between them. While both domains (for example `cljpn` and `html`) can have an entity called `body`, the addition of a namespace keeps the two identities distinct `cljpn:body` and `html:body`.

In XML and thus XBRL, everything is bound by a namespace, even if you don't see this. In that case a *default* (empty) namespace is used. Namespaces in XML are defined by a URI (Uniform Resource Identifier) and take often the form of a URL (Uniform Resource Locator), or otherwise a URN (Uniform Resource Name). So `http://www.uri.org/schema/value` or `urn:uri:org:schema:value` are both valid namespaces.

Because of the verbose nature of URIs they tend to be very long. To increase readability and ease of use it is possible to refer to a namespace by a **prefix**. The prefix is defined at the top of the schema file along with the namespace. Although the choice of the prefix is free, some prefixes are either a standard or universally accepted as the prefix for a given namespace. Some examples of these are:

prefix	namespace
xs	http://www.w3.org/2001/XMLSchema
link	http://www.xbrl.org/2003/linkbase
xlink	http://www.w3.org/1999/xlink
xbrli	http://www.xbrl.org/2003/instance
svg	http://www.w3.org/2000/svg

All elements (concepts) defined in a schema are defined within the namespace the schema dictates. This is done by the **targetNamespace** of the schema.
When we define the following concept

```
<xs:element abstract="false"
    id="cljpn_Assets"
    name="Assets"
    nillable="false"
    substitutionGroup="xbrli:item"
    type="xbrli:monetaryItemType"
    xbrli:balance="debit"
    xbrli:periodType="instant" />
```

In a schema file that has a targetNamespace defined like:

```
xmlns:cljpn="http://schemas.cooljapan.nl/taxonomy/cljpn"
```

and it has

```
targetNamespace="http://schemas.cooljapan.nl/taxonomy/cljpn"
```

then we must refer to this `assets` element as either (Clark notation):

```
{http://schemas.cooljapan.nl/taxonomy/cljpn}Assets
```

or the more common form with the prefix

```
cljpn:Assets
```

The latter form is also known as the QName or Qualified Name.

Whenever you see a colon (':') inside a tag name (`xs:element`), attribute name (`xbrli:balance`), attribute value (`xbrli:monetaryItemType`) or value, you are most likely dealing with a QName which consists of the prefix (of a namespace) and the local name of the element.

When writing a report, all prefixes and namespaces are provided by the taxonomy. There is not much to worry about for a filer, just be aware that namespaces exist and are heavily used in xbrl.

5 Units

When reporting a numeric value you also need to reference a unit. All non numerical datatypes must **not** reference a unit. A unit tells us the *of what* of the aspects. A monetary item type where the value represents a number in a specific currency, one must add a `ISO-4127 Currency code` as a Unit and reference this unit from the value. We will explore how this is done in the [XBRL Instance].

There are many types of units possible. The list of know units is published by the xBRL organisation. list of units and holds besides currencies the likes of *meters*, *celcius*, *joule* and many, many others.

Besides these there is also a registry which holds proposed units which ultimately will end up in the unit list Unit registry.

While financial data will often have currencies as their units, some need to be mentioned as well. The `pure`-unit for numbers that are just that. And the `shares`-unit as the unit of ownership in stock.

Units in xBRL Instances are mentioned on their own, just once for each used unit. They are referenced by their *id*.

```
<unit id="EUR">
  <measure>iso4217:EUR</measure>
</unit>
```

example unit definition for the currency Euro.

A value can have an optional *decimal*- and *precision*-attribute.

6 Contexts

We have seen that facts are made up of values and aspects. One of those aspects, namely the concept, is what gives the value its tag. The other aspects together form the context of the fact.

Contexts are written at the top of a XBRL instance. If we think back of the first fact I presented in the [Facts] chapter

```
Value: 1079,5
Concept: Net revenue
Entity: Cooljapan
Period: 2024
unit: Euro
decimals: -6
```

We will end up with a context that consists of the period and the entity. The concept will be the tag name of the value, and the unit and decimals are attributes on the value.

When facts have more aspects than the basic mentioned so far, they will also end up in a context. These extra aspects will come from dimensions and members. In the second part we will introduce them to you.

With the basics in place facts, concepts, and aspects we have the building blocks to write this fact in an XBRL instance. In the next chapter we have a look at our first instance document.

7 XBRL Instance

When writing a report we usually do so in XML. Nowadays it is also possible to write it in JSON or CSV. The notations differ, but the idea is the same, they are just representational different, the stated facts are off course the same. For brevity I'll only show the XML variant. The XML Variant consists of three parts.

- First we enumerate the different contexts that are used by the facts
- The unit(s) used by the facts
- The values

First we need to determine which report we are going to file. We will choose for our example an existing SBR taxonomy: KVK Micro taxonomy viewer

We will report a net revenue of 1000 euro for the year 2023

```
<?xml version="1.0" encoding="UTF-8"?>
<xbrli:xbrl xml:lang="nl"
  xmlns:iso4217="http://www.xbrl.org/2003/iso4217"
  xmlns:jenv-bw2-i="http://www.nltaxonomie.nl/nt19/jenv/20241211/dictionary/jenv-
↪  bw2-data"
  xmlns:link="http://www.xbrl.org/2003/linkbase"
  xmlns:xlink="http://www.w3.org/1999/xlink"
  xmlns:xbrli="http://www.xbrl.org/2003/instance">
  <link:schemaRef xlink:type="simple"
  xlink:href="http://www.nltaxonomie.nl/nt19/kvk/20241211/entrypoints/kvk-rpt-
↪  jaarverantwoording-2024-nlgaap-micro.xsd"/>
↪
  <xbrli:context id="context_1">
    <xbrli:entity>
      <xbrli:identifier
↪  scheme="http://www.kvk.nl/kvk-id">12345678</xbrli:identifier>
    </xbrli:entity>
    <xbrli:period>
      <xbrli:startDate>2022-01-01</xbrli:startDate>
      <xbrli:endDate>2022-12-31</xbrli:endDate>
    </xbrli:period>
  </xbrli:context>
```

```
  <xbrli:unit id="EUR">
    <xbrli:measure>iso4217:EUR</xbrli:measure>
  </xbrli:unit>
  <jenv-bw2-i:NetRevenue
      decimals="INF"
      contextRef="context_1"
      unitRef="EUR">1000</jenv-bw2-i:NetRevenue>
</xbrli:xbrl>
```

That might be a bit much to digest in full, let's look at this in detail.

- At the top we declare the required root node `xbrl` (with the prefix `xbrli` so the QName is `xbrli:xbrl`)
- on this node we define the optional `xml:lang` attribute and the namespaces with their prefixes used in this document.
- Next we refer to the taxonomy entrypoint which we used to create this instance. This is done via the `link:schemaRef` tag.

With the administration done we can create the three body parts.

- There is one context, which is made up of the entity aspect and the time aspect. We give it an id of `context_1`
- We declare that our filing is done in Euro's and give the currency the `id` of `EUR`
- We report a value of 1000 on the concept `NetRevenu` from the prefix `jenv-bw2-i`

With these three parts done we can close the document.

This is a very simple, but valid instance, apart from some required fields missing. The instances get a bit more complicated on the context level as we add extra 'dimensions' or 'axis'. But in principle this is all you need to know about an instance. The only difference with real world examples is the amount of facts.

The three main parts of an instance are: Contexts which have an ID attribute and an entity and period inside, unit(s), also have their ID attribute and values for concepts with reference to a context and an unit.

In brief so far:

- Reports consist of Facts,
- Facts consist of aspects and values,
- Concepts are the main aspect of a Fact.
- Numerical values must reference a unit,
- The other aspects make up a context which is reference by a fact.

Apart from the optional dimensions and members which we will cover in the next chapter, this is all there is to an XBRL instance. A complete filing will of course have a lot of contexts and way more values in there, but that's all just repetition. And again, if you are creating a complete filing there must be software available to help you create the XBRL.

8 Adding aspects

We started our introduction to xbrl with facts. And we have learned that a fact can be presented as a value for a concept (from a taxonomy) accompanied by an entity (who) a period or duration and, when dealing with numerical facts a unit. With these aspects a lot of facts can be stated. But most of the time there need to be more aspects to a fact. The toy factory *Cooljapan* does its internal reporting with xbrl. And they want to report on the colours of the toys produced. A simple fact table for such a report could then look like this

concept	entity	unit	period	colour	value
yoyo	Cooljapan	pure	2024	red	1000
yoyo	Cooljapan	pure	2024	yellow	2000
yoyo	Cooljapan	pure	2024	blue	750
yoyo	Cooljapan	pure	2024	total	3750
toll	Cooljapan	pure	2024	red	500
toll	Cooljapan	pure	2024	yellow	500
toll	Cooljapan	pure	2024	blue	500
toll	Cooljapan	pure	2024	total	1500

a fact table with two different concepts and a colour aspect with 4 options

The technical term for what we have added here is a **dimension** with 4 **members**. Each aspect that is not one of the default aspects (concept, entity, period, unit) is added as a member of a dimension. In the second part we'll learn how a dimension is created and how the taxonomy tells you when and where to add extra aspects. All that matters now is that we see that it is possible to create more detailed insights by adding aspects.

Now the fact table is only one way of presenting the facts in an instance. The table above could just as well be drawn with two rows and a heading stating the repetitious aspects of entity, period and unit.

company	period	unit
Cooljapan	2024	pure

	red	yellow	blue	total
yoyo	1000	2000	750	3750
toll	500	500	500	1500

table stating the same facts on two rows with 3 basic aspects mentioned in the heading

If these facts were to be reported in an instance, it could look like this:

```
<xbrli:context id="context_1">
    <xbrli:entity>
        <xbrli:identifier scheme="urn:cljpn:schema:entyti:id">
        cooljapan</xbrli:identifier>
    </xbrli:entity>
    <xbrli:period>
        <xbrli:startDate>2024-01-01</xbrli:startDate>
        <xbrli:endDate>2024-12-31</xbrli:endDate>
    </xbrli:period>
    <xbrli:scenario>
        <xbrldi:explicitMember dimension="cljpndim:colours">
            cljpnmem:red
        </xbrldi:explicitMember>
    </xbrli:scenario>
</xbrli:context>
<xbrli:context id="context_2">
    <xbrli:entity>
        <xbrli:identifier scheme="urn:cljpn:schema:entyti:id">
        cooljapan</xbrli:identifier>
    </xbrli:entity>
    <xbrli:period>
        <xbrli:startDate>2024-01-01</xbrli:startDate>
        <xbrli:endDate>2024-12-31</xbrli:endDate>
    </xbrli:period>
    <xbrli:scenario>
        <xbrldi:explicitMember dimension="cljpndim:colours">
            cljpnmem:yellow
        </xbrldi:explicitMember>
    </xbrli:scenario>
```

```
</xbrli:context>
<xbrli:context id="context_3">
    <xbrli:entity>
        <xbrli:identifier scheme="urn:cljpn:schema:entyti:id">
        cooljapan</xbrli:identifier>
    </xbrli:entity>
    <xbrli:period>
        <xbrli:startDate>2024-01-01</xbrli:startDate>
        <xbrli:endDate>2024-12-31</xbrli:endDate>
    </xbrli:period>
    <xbrli:scenario>
        <xbrldi:explicitMember dimension="cljpndim:colours">
            cljpnmem:blue
        </xbrldi:explicitMember>
    </xbrli:scenario>
</xbrli:context>
<xbrli:context id="context_4">
    <xbrli:entity>
        <xbrli:identifier scheme="urn:cljpn:schema:entyti:id">
        cooljapan</xbrli:identifier>
    </xbrli:entity>
    <xbrli:period>
        <xbrli:startDate>2024-01-01</xbrli:startDate>
        <xbrli:endDate>2024-12-31</xbrli:endDate>
    </xbrli:period>
    <xbrli:scenario>
        <xbrldi:explicitMember dimension="cljpndim:colours">
            cljpnmem:allcolours
        </xbrldi:explicitMember>
    </xbrli:scenario>
</xbrli:context>

<xbrli:unit id="PURE">
    <xbrli:measure>xbrli:pure</xbrli:measure>
</xbrli:unit>

<cljpn:yoyoAmount
    decimals="INF"
    contextRef="context_1"
    unitRef="PURE">1000</cljpn:yoyoAmount>
<cljpn:yoyoAmount
    decimals="INF"
    contextRef="context_2"
    unitRef="PURE">2000</cljpn:yoyoAmount>
<cljpn:yoyoAmount
    decimals="INF"
```

```
    contextRef="context_3"
    unitRef="PURE">750</cljpn:yoyoAmount>
<cljpn:yoyoAmount
    decimals="INF"
    contextRef="context_4"
    unitRef="PURE">3750</cljpn:yoyoAmount>
<cljpn:tollAmount
    decimals="INF"
    contextRef="context_1"
    unitRef="PURE">500</cljpn:tollAmount>
<cljpn:tollAmount
    decimals="INF"
    contextRef="context_2"
    unitRef="PURE">500</cljpn:tollAmount>
<cljpn:tollAmount
    decimals="INF"
    contextRef="context_3"
    unitRef="PURE">500</cljpn:tollAmount>
<cljpn:tollAmount
    decimals="INF"
    contextRef="context_4"
    unitRef="PURE">1500</cljpn:tollAmount>
```

part of an xbrl instance stating the facts from the previous tables

The three parts are hopefully recognisable; the contexts (4 this time, and slightly more complex), the unit, this time not a currency, but the `xbrli:pure` unit which should be used for plain numbers. And 8 values make up the third part. Starting with the last ones, we saw in both tables that we have two types of toys and from the in published taxonomy we find two concepts `yoyoAmount` and `tollAmount` both in the namespace with the prefix `cljpn`. Both concepts occur four times in the instance, each time referencing a different context.

Now let us examine the contexts for example `context_2` here repeated for clarity.

```
<xbrli:context
    id="context_2">
  <xbrli:entity>
    <xbrli:identifier
        scheme="urn:cljpn:schema:entity:id">cooljapan
    </xbrli:identifier>
  </xbrli:entity>
  <xbrli:period>
    <xbrli:startDate>2024-01-01</xbrli:startDate>
    <xbrli:endDate>2024-12-31</xbrli:endDate>
```

```
  </xbrli:period>
  <xbrli:scenario>
    <xbrldi:explicitMember
        dimension="cljpndim:colours">cljpnmem:yellow
    </xbrldi:explicitMember>
  </xbrli:scenario>
</xbrli:context>
```

We start just like the previous example, with an entity, followed by a period. Then there is this new tag `xbrli:scenario`. This section holds the dimensions (or axis). The tag could also be `xbrli:segment`, which functions exactly the same. The choice wether to use *segment* or *scenario* is dictated by the taxonomy. How is explained in the second part where we dive into the taxonomy. They both exist for historical reasons but their functionality has been superseded by dimensions.

The tag is followed by yet another new tag `xbrldt:explicitMember`. This tag has one required tag `dimension`. The value of that attribute is a QName that points to a concept in the taxonomy. This concept is also the top of our colours dimension. We've seen that we report on three real colours and a total. So this dimension has four members that can also be found in the taxonomy. These members are also concepts. We see that the value is `cljpnmem:yellow`, and we saw that in the table the heading was just 'yellow'. That value comes from a label tied to the concept. In the report we must point to the concept by it's QName.

Besides an *explicitMember* there is also `xbrldt:typedMember`, and in contrast to *scenario* and *segment* which have no different meaning, a *typedMember* has. In our colours example we've unknowingly created an *explicit dimension*. It means that we've created a dimension whose members can be stated before hand. Colours, types of drinks, types of material, classes of equity, countries, etc. are examples of dimensions that could be created as explicit dimensions. It's possible to state all possibilities in the taxonomy so the filer can choose which ones can reported on.

A typed dimension holds members that can not be known beforehand when writing the taxonomy, or it would be a too great effort and meaningless to add all the possibilities as a member. Suppose we want to report per telephone number. In theory it could be possible to add all possible phone numbers to the taxonomy. But it would be madness. It is also quite common to report on individuals tied to a company. To identify this person one usually works with a name. Now to state all possible names inside a taxonomy is impossible. For these cases one uses a typed dimension. The member can be 'any value that fits within the type definition'. And a type definition could be `string(254)`.

For the colours we have created a dimension 'Colours' with four members. But Cooljapan has invested on new machinery that can handle an unlimited variety of colours. So we change our original explicit dimension into a typed dimension. The type is a `string(254)`. So when we want to report the same values on the same colours, the contexts will look slightly different.

```
<xbrli:context
    id="context_2">
  <xbrli:entity>
    <xbrli:identifier
        scheme="urn:cljpn:schema:entity:id">cooljapan
    </xbrli:identifier>
  </xbrli:entity>
  <xbrli:period>
    <xbrli:startDate>2024-01-01</xbrli:startDate>
    <xbrli:endDate>2024-12-31</xbrli:endDate>
  </xbrli:period>
  <xbrli:scenario>
    <xbrldi:typedMember
        dimension="cljpndim:colours">
        <cljpnmem:colour>Crimson red</cljpnmem:colour>
    </xbrldi:typedMember>
  </xbrli:scenario>
</xbrli:context>
```

Inside the `xbrli:scenario` we see that we add a `typeMember` from the colours-dimension. The taxonomy will have one member defined in this dimension. The name of that member is used as the tag. The actual colour is the value for this member.

At this point we have seen all the parts that make an XBRL instance. All that remains to cover is called taxonomy. It is the datamodel and explanation of the instance that can be created with it. The concepts, the dimensions and the units you can use to report are all defined in the taxonomy.

9 Taxonomy

So far I've only introduced the building blocks of an instance or report. Yet the term *taxonomy* has been used a couple of times. A taxonomy is the complete definition of one or more filings. An instance is only possible if there exists a taxonomy on which the report is based. To create a report one must first *read* the taxonomy on which you want to base the report. Only then you have knowledge of which concepts can/must be reported on and how. In the mini-instance we created in [XBRL Instance] we see a line:

```
<link:schemaRef xlink:type="simple"
↪  xlink:href="http://www.nltaxonomie.nl/nt19/kvk/20241211/entrypoints/kvk-rpt-
↪  jaarverantwoording-2024-nlgaap-micro.xsd"/>
```

Here we state that we want to file a report against this entrypoint. An entrypoint is the first file you, or more often, your software opens when you want to discover a taxonomy.

An xbrl taxonomy consists of at least one xml schema file , which can be the starting point of a long line of subsequent imports of other schema files or linkbase files which are referenced by means of the xbrl specifications.

One of the key aspects of xbrl is that it not only provides the syntax and the data definitions, but also adds semantic meaning to this definition. Where syntax means that the taxonomy describes how and what you should report. Semantics on the other hand adds meaning to the facts and how these are related.

This is done by building hierarchies by combining different building blocks of the taxonomy together into meaningful groups.

A small but important hierarchy is formed around the Concept itself. I've already mentioned that a Concept can have labels, documentation and references. These little hierarchies are a good starting point to look at a taxonomy and how things get linked. But first we look at how these little hierarchies are added to the taxonomy and how we can read a taxonomy.

To start the discovery of a taxonomy one reads the entrypoint!

10 Entrypoints and Linkbases

From this point on there will be a lot more xml than in the first part. Remember xml is only the tool used to express xbrl. It's 'just' notation which can be interpreted by well programmed machines that have knowledge of the xbrl specifications. The xbrl organisation has a compliance suite to which all certified xbrl software have to adhere.

To create a filing (an xbrl instance) one references an entrypoint or schema file by means of a `link:schemaRef`, for instance

```
<link:schemaRef
    xlink:type="simple"
    xlink:href="http://www.nltaxonomie.nl/nt19/kvk/20241211/entrypoints/kvk-rpt-
↪   jaarverantwoording-2024-nlgaap-micro.xsd"/>
```

The file that is referenced via the `xlink:href` is called an entrypoint. An entrypoint is a view on a taxonomy (or Discoverable Taxonomy Set -DTS-). As we will see we can create different hierarchies that give structure and meaning to the concepts in a taxonomy. Taxonomies can consist of more than just one filing or report. By creating different files with different parts of the taxonomy definitions a taxonomy writer is able to create different reports with the ability to re-use general parts of the different filings. These files are either more xml schema files (`.xsd`) or so called `linkbase` files.

A view or report starts with an entrypoint, which is always an xml-schema. At the top we have some namespace declarations, one xbrl specific. `xmlns:link="http://www.xbrl.org/2003/linkbase"` which is used to include all the different parts that create the specific filing. Remember this is a namespace, there is no file hosted on the URL[1].

In xbrl we use the 'freeform' `xs:annotation`, `xs:appinfo`[2] to enumerate the linkbases used. Now let's write this out in xml

[1] xbrl.org does host a file on that location to point at the schema file, which can be found here: ttp://www.xbrl.org/2003/xbrl-linkbase-2003-12-31.xsd

[2] see https://docstore.mik.ua/orelly/xml/schema/ch14_02.htm for more examples of these elements

```
<xs:schema
    xmlns:kvk-rpt-h-u="http://www.nltaxonomie.nl/nt19/kvk/20241211/entrypoints/kvk-
↪   rpt-jaarverantwoording-2024-nlgaap-micro"
    xmlns:link="http://www.xbrl.org/2003/linkbase"
    xmlns:xlink="http://www.w3.org/1999/xlink"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    attributeFormDefault="unqualified"
    elementFormDefault="qualified"
    id="kvk-rpt-h-u"
       targetNamespace="http://www.nltaxonomie.nl/nt19/kvk/20241211/entrypoints/kvk-
↪   rpt-jaarverantwoording-2024-nlgaap-micro">
<xs:annotation>
  <xs:appinfo>
    <link:linkbaseRef
      xlink:arcrole="http://www.w3.org/1999/xlink/properties/linkbase"
      xlink:href="../validation/kvk-balance-sheet-lineitems_u-def.xml"
      xlink:role="http://www.xbrl.org/2003/role/definitionLinkbaseRef"
      xlink:type="simple"/>
```

Inside the `xs:appinfo` we see the link to the definition of the line-items of the balance sheet. This particular entrypoint has 137 different linkbases that need to be loaded and examined to have a complete overview of the report at hand.

As we look closer to the actual import we see the attributes `xlink:arcrole` and `xlink:role` The arcrole points to the standard xlink properties of a linkbase. The role can have any URI as a value, but there are just a couple of standard roles defined by xbrl.org. These are:

- **Presentation Linkbase**: Used to define the structure and order in which concepts are presented in financial statements.
- **Calculation Linkbase**: Used to define the arithmetic relationships between concepts, such as summation.
- **Definition Linkbase**: Used to define more complex relationships between concepts, like dimension relationships.
- **Label Linkbase**: Used to provide human-readable labels for concepts, including different types of labels like terse, verbose, and documentation labels.

Each of these will be studied in more detail in the following chapters. For the moment it's enough to understand that the files mentioned in the `xlink:href` attributes must be loaded and parsed by the software. When all the files are loaded and the parsing is done, the software holds the definition of a specific filing, defined by the entrypoint.

Besides loading linkbases it is also possible to import or include other schema files. This can also be external taxonomies defined by a regulatory instance. For instance a country or a currency taxonomy

often get imported from an authoritative source.

We now start our discovery by looking at a label linkbase.

11 Labels, documentation and references

11.1 Labels

Entrypoints can load a linkbase. In fact all xml schema files can load a linkbase file by the same means shown with the entrypoint; *xs:annotation* and *xs:appinfo*. In xbrl we do this by means of a `link:linkbaseRef`, which comes from the xbrl-standard. One of the standard linkbases is the label linkbase.

We have seen that concepts are the *what* of what gets reported. Because most xbrl reports are based on regulation they thought of a way to add semantic information to a concept, such as the regulation the concept is based on, documentation and labels. All this information can be stored inside the taxonomy, in multiple languages if needed.

We start by adding labels to a concept, for example:

```
<xs:element
    abstract="false"
    id="nl-cd_POBoxNumber"
    name="POBoxNumber"
    nillable="false"
    substitutionGroup="xbrli:item"
    type="nl-types:nonNegative20ItemType"
    xbrli:periodType="duration"/>
```

And the labels themselves in both Dutch and English to this concept.

```
<link:label
    id="nl-cd_POBoxNumber_label_nl"
    xlink:label="nl-cd_POBoxNumber_label_nl"
    xlink:role="http://www.xbrl.org/2003/role/label"
    xlink:type="resource"
    xml:lang="nl">Postbusnummer</link:label>
<link:label
    id="nl-cd_POBoxNumber_label_en"
    xlink:label="nl-cd_POBoxNumber_label_en"
```

```
    xlink:role="http://www.xbrl.org/2003/role/label"
    xlink:type="resource"
    xml:lang="en">PO box number</link:label>
```

These two labels, created in the *link* namespace carry information in their attributes. The `xml:lang` tells us which language, the `xlink:role` attribute tells us these are standard labels. The `xlink:role` attribute for labels can have different values. Some common values are `totalLabel`, `periodEndLabel`, `terseLabel`. For now we'll focus on the standard label and link them to the PO Box concept.

For this linking XBRL uses Arcs and Locators. Arcs associates objects to another, and do so with the role of the arc, which is called `arcrole`. To link concepts to labels we use the `concept-label`-arcrole.

```
<link:labelArc
    xlink:arcrole="http://www.xbrl.org/2003/arcrole/concept-label"
    xlink:from="nl-cd_POBoxNumber_loc"
    xlink:to="nl-cd_POBoxNumber_label_nl"
    xlink:type="arc"/>
<link:labelArc
    xlink:arcrole="http://www.xbrl.org/2003/arcrole/concept-label"
    xlink:from="nl-cd_POBoxNumber_loc"
    xlink:to="nl-cd_POBoxNumber_label_en"
    xlink:type="arc"/>
```

An arc always consists of the following attributes:

- `xlink:arcrole` Which role does this relation have?
- `xlink:from` The source of the relationship
- `xlink:to` The target of the relationship
- `xlink:type` `arc`

In this case, the `xlink:from` is for both labels the same and reads: `nl-cd_POBoxNumber_loc`. Following the specification, this `xlink:from` points to a `label` of a node of the type `locator`. Here we see the two locators that point to the concept via the `xlink:href` and are 'found' by the arc on the value of the `xlink:label`. Because we are adding both a Dutch and English label to the same concept it is understandable both locators point to the same concept in the `xlink:hraf`.

```
<link:loc
    xlink:href="nl-common-data.xsd#nl-cd_POBoxNumber"
    xlink:label="nl-cd_POBoxNumber_loc"
```

```
    xlink:type="locator"/>
<link:loc
    xlink:href="nl-common-data.xsd#nl-cd_POBoxNumber"
    xlink:label="nl-cd_POBoxNumber_loc"
    xlink:type="locator"/>
```

A locator always must have the following attributes

- `xlink:href` Where does the locator point to? This in itself is a URL with an ID-Anchor. in this case `nl-common-data.xsd` as the (relative) URL and `nl-cd_POBoxNumber` as the ID of the element to be located in that file. In this case the locater points to a concept, but it could also point to other resources.
- `xlink:label` The anchor the arc attaches to.
- `xlink:type` `locator`

11.2 Roles

In the above example we have discussed one Concept having two labels attached via arcs. These arcs have the tag `link:labelArc`. One of the mandatory attributes is the `xlink:arcrole` which in this case points to the URN `http://www.xbrl.org/2003/arcrole/concept-label`. As I said, this is a URN, that looks like an URL. But on that location no file is hosted. You can however see a couple of the standard roles defined by the xbrl organization at this URL: http://www.xbrl.org/2003/xbrl-role-2003-07-31.xsd As well as an Arc, a label definition can have a role as well. These roles determine what label is displayed. Under certain circumstances a concept can represent a summation of underlying concepts in an hierarchy. In that case the taxonomy writer will assign a different role to the label, in that case `http://www.xbrl.org/2003/role/totalLabel`. In the document linked above you can view all the roles the xbrl standard recognizes.

11.3 Label linkbase file

We now have the building blocks to construct a complete linkbase holding these labels and their arcroles and locators. To construct the complete file we need to add one more tag at the top of the document to hold all the labels, arcs and locators together. In this case the `link:labelLink`. Different types of linkbases have different tags in the `link` namespace as we shall see later on. The start of the file would look like this:

```
<?xml version="1.0" encoding="UTF-8" ?>
<link:linkbase
    xmlns:link="http://www.xbrl.org/2003/linkbase"
    xmlns:xlink="http://www.w3.org/1999/xlink">
  <link:labelLink
      xlink:role="http://www.xbrl.org/2003/role/link"
      xlink:type="extended">
    <link:label
       id="kvk-abstr_EntityAddressTitle_label_en"
       xlink:label="kvk-abstr_EntityAddressTitle_label_en"
       xlink:role="http://www.xbrl.org/2003/role/label"
       xlink:type="resource"
       xml:lang="en">Address of the legal entity
    </link:label>
    <link:labelArc
       xlink:arcrole="http://www.xbrl.org/2003/arcrole/concept-label"
       xlink:from="kvk-abstr_EntityAddressTitle_loc"
       xlink:to="kvk-abstr_EntityAddressTitle_label_en"
       xlink:type="arc"/>
    <link:loc
        xlink:href="kvk-abstracts.xsd#kvk-abstr_EntityAddressTitle"
        xlink:label="kvk-abstr_EntityAddressTitle_loc"
        xlink:type="locator"/>
```

11.4 Documentation

A special label role is the documentation role. `http://www.xbrl.org/2003/role/documentation`. Like any other label it can be in different languages. In these labels the taxonomy writer can add helpful information to explain what the concept represents and what is expected of the filer. Documentation labels are identical to normal labels in every aspect, It's just that most software treats these labels differently and they are mentioned in the specifications. They also get defined within the same linkbase as labels are. Apart from the role they are in no way different from normal labels.

11.5 References

The third (or technically second) way to add information to a concept is by means of a reference. Because most taxonomies are based on legislation, standards or some rulebooks, taxonomy writers need a way to reference a law or a bill which gives clarification from the legal side. References can include a broad range of sources. This could be books, magazines, websites, articles, etc.

The PO Box number concept has a reference in the taxonomy. In this case it only points to the standard: `NEN5825`, without further information or a link. Lets see how a reference is defined in xml:

```
<link:reference
    id="nl-cd_NEN5825_2002_ref"
    xlink:label="nl-cd_NEN5825_2002_ref"
    xlink:role="http://www.xbrl.org/2003/role/reference"
    xlink:type="resource">
  <ref:IssueDate>2002</ref:IssueDate>
  <ref:Name>NEN5825</ref:Name>
</link:reference>
```

The reference has its own tag `link:reference` Attributes:

- `xlink:role` : `http://www.xbrl.org/2003/role/reference`
- `xlink:type` : `reference`
- `id` : optional, unique
- `xlink:label` unique value (like id)

There are many tags which can be used inside an `link:reference` node. In the example above you see two of them: `IssueDate` and `Name`. But there are many others that can be added like `pages`, `clause`, `paragraph`, `publisher` to name a few[1].

These references get linked to a concept in the same way we add labels to concepts. By means of arcs and locators. Remember, the arc gives meaning to the relationship by means of its role. The locator points to the intended resource. Lets look at the specific arc for a reference:

```
<link:referenceArc
    xlink:arcrole="http://www.xbrl.org/2003/arcrole/concept-reference"
    xlink:from="nl-cd_POBoxNumber_loc"
    xlink:to="nl-cd_NEN5825_2002_ref"
    xlink:type="arc"/>
```

This time we see that this arc has a tag `link:referenceArc` and an arcrole `http://www.xbrl.org/2003/arcro` which tells us that we deal with a concept-reference relationship. The other attributes we have seen before, the `xlink:from`, `xlink:to` and `xlink:type`. The `from` side has to be resolved via a locator. The `to` side points to the `link:reference` from the previous xml example. The locator for completeness:

[1]The xbrl organisation recognises these values as valid parts: *name, number, issuedate, chapter, article, note, section, subsection, publisher, paragraph, subparagraph, clause, subclause, appendix, example, page, exhibit, footnote, sentence, uri, uridate http://www.xbrl.org/2006/ref-2006-02-27.xsd

```
<link:loc
    xlink:href="nl-common-data.xsd#nl-cd_POBoxNumber"
    xlink:label="nl-cd_POBoxNumber_loc"
    xlink:type="locator"/>
```

Again points the `xlink:href` to the POBoxNumber concept and the `xlink:label` attribute corresponds with the `xlink:from` from the referenceArc.

To create a reference linkbase holding these references and their arcs and locators together we add a top level linkbase tag and a `link:referenceLink` tag to a file.

```
<?xml version="1.0" encoding="UTF-8"?>
<link:linkbase
    xmlns:link="http://www.xbrl.org/2003/linkbase"
    xmlns:ref="http://www.xbrl.org/2006/ref"
    xmlns:xlink="http://www.w3.org/1999/xlink"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.xbrl.org/2006/ref
 ↪  http://www.xbrl.org/2006/ref-2006-02-27.xsd">
  <link:referenceLink
      xlink:role="http://www.xbrl.org/2003/role/link"
      xlink:type="extended">
  <link:reference
      id="nl-cd_NEN5825_2002_ref"
      xlink:label="nl-cd_NEN5825_2002_ref"
      xlink:role="http://www.xbrl.org/2003/role/reference"
      xlink:type="resource">
  <ref:IssueDate>2002</ref:IssueDate>
  <ref:Name>NEN5825</ref:Name>
</link:reference>
<link:referenceArc
    xlink:arcrole="http://www.xbrl.org/2003/arcrole/concept-reference"
    xlink:from="nl-cd_POBoxNumber_loc" xlink:to="nl-cd_NEN5825_2002_ref"
    xlink:type="arc"/>
<link:loc
    xlink:href="nl-common-data.xsd#nl-cd_POBoxNumber"
    xlink:label="nl-cd_POBoxNumber_loc"
    xlink:type="locator"/>
```

11.6 A general Mechanism Arcs and Locators

With the mechanism of arc and locators we are able to create all kinds of hierarchies. Here shown with the introduction of labels and references bundled in linkbase files. In this part we created a hier-

archy with a concept on top and different labels and references as children. The variety of labels is the multiplication of possible label roles and desired languages. So if you want to have 4 languages and use 5 different label roles, a concept can have as much as 20 labels attached. In theory the number of references can be unlimited. The next type of standard linkbase we will explore is the presentation network.

12 Presentation, abstract Concepts and Linkroles

In the previous chapter we've seen how we can add labels and references to a concept to provide extra information to the filers. Taxonomy writers have more tools to add structure to the taxonomy and to create logical connected parts. One of those ways to add structure is by means of a *presentation linkbase*. Linking of this information is done in the same way as labels and references, by means of arcs and locators.

The presentation linkbase is mainly a readability help for humans. The presentation linkbase gives taxonomy writers the ability to add hierarchy to the concepts. For machines it does not really matter that the PO Box number and Total assets are on the same level in an instance, for humans it's important to have structure.

Here we add presentational structure, we do this by layering the concepts into logical connected parts. For instance the PO Box number is most likely part of an address. To create a hierarchy with all address concepts we need to introduce the 'abstract concept' or plain abstract as I prefer to call them in contrast to concepts that can hold values. An abstract concept differs from a concept:

- has `abstract=true`
- brings structure to a taxonomy but can hold no values (so they won't ever appear in an instance)
- must not have a datatype nor balance.

With these rules we can introduce a concept to our taxonomy which will be the top of an address.

```
<xs:element
    abstract="true"
    id="kvk-abstr_EntityAddressTitle"
    name="EntityAddressTitle"
    nillable="false"
    substitutionGroup="sbr:presentationItem"
    type="xbrli:stringItemType"
    xbrli:periodType="duration"/>
```

This concept can of course, by nature, have labels attached.

```
<link:label
    id="kvk-abstr_EntityAddressTitle_label_en"
    xlink:label="kvk-abstr_EntityAddressTitle_label_en"
    xlink:role="http://www.xbrl.org/2003/role/label"
    xlink:type="resource"
    xml:lang="en">Address of the legal entity [abstract]
</link:label>
<link:labelArc
    xlink:arcrole="http://www.xbrl.org/2003/arcrole/concept-label"
    xlink:from="kvk-abstr_EntityAddressTitle_loc"
    xlink:to="kvk-abstr_EntityAddressTitle_label_en"
    xlink:type="arc"/>
<link:loc
    xlink:href="kvk-abstracts.xsd#kvk-abstr_EntityAddressTitle"
    xlink:label="kvk-abstr_EntityAddressTitle_loc"
    xlink:type="locator"/>
```

xml for the English label of this abstract

This concept is going to be the top of the hierarchy which has the title `Address of the legal entity [abstrac` It is custom to add the `[abstact]` to the label to give the reader an extra clue that this is a grouping element which does not represent any data-point in an instance. This is however not an obligation. For instance in the Dutch taxonomy (NT) from which I plucked this example this extra tag is absent.

To create a presentation hierarchy we need to write another linkbase file, just as for the labels. For this linkbase we will introduce the `link:presentationLink` and the `link:roleRef` the latter references the definition of this particular presentation hierarchy. We are describing a part of a hierarchy, which can be looked at in full online.

- Address of the legal entity abstract
  - …
  - PO Box number
  - Postal Code NL
  - Place of residence NL
  - …

By adding an abstract on top of these concepts we are able to visually group the concepts that make up the legal entity address. At the top of this linkbase we see the following

```
<link:linkbase
    xmlns:link="http://www.xbrl.org/2003/linkbase"
    xmlns:xlink="http://www.w3.org/1999/xlink">
```

```
  <link:roleRef
      roleURI="urn:kvk:linkrole:entity-address"
      xlink:href="../dictionary/kvk-linkroles.xsd#kvk-lr_EntityAddress"
      xlink:type="simple"/>
  <link:presentationLink
      xlink:role="urn:kvk:linkrole:entity-address"
      xlink:type="extended">
```

The first unknown tag is the `link:roleRef`, which defines a `roleURI` and links to a definition in a schema file. The definition inside this `kvk-linkroles.xsd` file is as follows:

```
<link:roleType
    id="kvk-lr_EntityAddress"
    roleURI="urn:kvk:linkrole:entity-address">
  <link:definition>Adres van de rechtspersoon</link:definition>
  <link:usedOn>gen:link</link:usedOn>
  <link:usedOn>link:definitionLink</link:usedOn>
  <link:usedOn>link:presentationLink</link:usedOn>
</link:roleType>
```

The best way to look at a linkrole is that it's the top of a hierarchy, it binds different hierarchies with aspects in common. The definition shown here shows us that the linkrole (`link:roleType`) has an `id`, which is used to locate the linkrole and a `roleURI`. This URI is what binds all hierarchies together. Furthermore we see that we can add a definition to the linkrole by means of `link:definition` and we see three instances of `link:usedOn`. These values represent the different types of hierarchies this linkrole can participate in. And as we can see, this specific linkrole can be used on a `link:presentationLink`, as well as on generic links (for example tables) and definition links. We will talk about these later.

After the `link:roleRef` we see another new tag; `link:presentationLink`. We see two attributes

- `xlink:role`: which should be a roleURI of a linkrole
- `xlink:type`: `extended` for presentationLink

What follows is similar to what we've seen with the labels. An arc and a locator to link Concepts to this presentation. I'll repeat the top of the xml for completeness.

```
<?xml version="1.0" encoding="UTF-8"?>
<link:linkbase
    xmlns:link="http://www.xbrl.org/2003/linkbase"
    xmlns:xlink="http://www.w3.org/1999/xlink">
```

```
  <link:roleRef
      roleURI="urn:kvk:linkrole:entity-address"
      xlink:href="../dictionary/kvk-linkroles.xsd#kvk-lr_EntityAddress"
      xlink:type="simple"/>
  <link:presentationLink
      xlink:role="urn:kvk:linkrole:entity-address"
      xlink:type="extended">
    <link:loc
        xlink:href="kvk-abstracts.xsd#kvk-abstr_EntityAddressTitle"
        xlink:label="kvk-abstr_EntityAddressTitle_loc"
        xlink:type="locator"/>
    <link:loc
        xlink:href="nl-common-data.xsd#nl-cd_POBoxNumber"
        xlink:label="nl-cd_POBoxNumber_loc"
        xlink:type="locator"/>
    <link:loc
        xlink:href="nl-common-data.xsd#nl-cd_PostalCodeNL"
        xlink:label="nl-cd_PostalCodeNL_loc"
        xlink:type="locator"/>
    <link:loc
        xlink:href="nl-common-data.xsd#nl-cd_PlaceOfResidenceNL"
        xlink:label="nl-cd_PlaceOfResidenceNL_loc"
        xlink:type="locator"/>

    <link:presentationArc
        order="5"
        xlink:arcrole="http://www.xbrl.org/2003/arcrole/parent-child"
        xlink:from="kvk-abstr_EntityAddressTitle_loc"
        xlink:to="nl-cd_POBoxNumber_loc"
        xlink:type="arc"/>
    <link:presentationArc
        order="6"
        xlink:arcrole="http://www.xbrl.org/2003/arcrole/parent-child"
        xlink:from="kvk-abstr_EntityAddressTitle_loc"
        xlink:to="nl-cd_PostalCodeNL_loc"
        xlink:type="arc"/>
    <link:presentationArc
        order="7"
        xlink:arcrole="http://www.xbrl.org/2003/arcrole/parent-child"
        xlink:from="kvk-abstr_EntityAddressTitle_loc"
        xlink:to="nl-cd_PlaceOfResidenceNL_loc"
        xlink:type="arc"/>
  </link:presentationLink>
</link:linkbase>
```

The first concept we encounter is the abstract title we've added to the taxonomy at the start of this

chapter. Then the other three concepts we wanted in this hierarchy are found by their locators. Next we start to interpret the `link:presentationArc`. We see again like the arcs with the labels, we have a `from` and `to` attribute and an arcrole. The value for a presentationArc is always `http://www.xbrl.org/2003/arcrole/parent-child`. That's why some call this hierarchy also the 'parent-child network'. When we look at the `xlink:from` on these arcs, we see that they all have the same value, namely the locator for the abstract Concept that is the title of this address hierarchy. And when we use the standard labels instead of the concept name we get the hierarchy from earlier

```
+  Address of the legal entity [abstract]
    +  PO Box number
    +  Postal Code NL
    +  Place of residence NL
```

There is also an optional `order` on each presentationArc. With this you can tell the rendering which concept should follow which. If you omit this attribute, the order is not guarantied.

It is possible for children to be a parent as well. So you can nest hierarchies as deep as you want. A presentation hierarchy of a small income statement could look as follows

```
+  Income statement
    +  Total of income (totalLabel)
        +  Net revenue
        +  Other operating income
        +  Other income
    +  Total of expenses (negatedTotalLabel)
        +  Costs of raw materials and consumables (negatedLabel)
        +  Wages (negatedLabel)
        +  Depreciation, amortisation and decrease in value of assets (negatedLabel)
        +  Other operating expenses (negatedLabel)
        +  Other expenses (negatedLabel)
    +  Total of result before tax (totalLabel)
    +  Income tax expense (negatedLabel)
    +  Total of result after tax (totalLabel)
```

In this example we see between brackets sometimes label names. It is possible to add a `preferredLabel` attribute to the presentationArc. The value of this attribute should then be used to select the appropriate label for the concept which is the child of the presentationArc. The value must be a valid URI for the labelRole. For instance

```
preferredLabel="http://www.xbrl.org/2003/role/totalLabel"
```

Where we tell that we want the label associated to the concept that has that particular label-role when displaying the concept in this presentation. As we've seen in the previous chapter the amount of labels can grow quite quickly when you have more languages you need to cover and have concepts that need different labels in different presentations. You see in this particular case that the taxonomy designer choose for a negated label for every concept that denotes a credit concept. That is a choice a designer can make. If we look at the definition of that concept, we will see that the standard label of *wages* is exactly the same *wages*. In this case the designer decided that the negated label roles add meaning to the concept, you can also deduct by the label role that the concept is *credit*.

13 Dimensions, domains and members

When the default aspects aren't enough fine-grained to report a fact, we can add one or more dimensions to the fact to give more context. When we were [Adding aspects] we came to know the toy factory that produces coloured toys. We can report on the amount of *yoyo*'s we produce with the default aspects. We have a concept `toys` which is of type `integer` and has a unitref of `pure`.

```
Concept: yoyoAmount
Entity: Cooljapan
Period: 2025-10-10
Value: 10
```

Management has decided that we have to report on each colour of these yoyo's. To be able to do this, we add some elements to our taxonomy. At the top we place something we refer to as dimension, but you can also think of it as an axis of a table. For this example we create a concept called `cljpndim:colours` with a label `Colours`. We will add this as an Abstract to our taxonomy. A dimensional concept accepts no values.

The dimension can have one or more children, which are called *members*. It is possible for members to have children as well. If that is the case, the topmost member (the one directly linked to the dimension) is called a *domain*. To get back to our example, we would like to report on the colours, *yellow*, *blue* and *red*. This means we have to add three members to the *colours* dimension. We start by defining these individual colours as a concept `cljpnmem:red` `cljpnmem:yellow` and `cljpn:blue` with the labels `red`, `yellow` and `blue` respectively.

We now have the tools to report on these colours

	Red	Yellow	Blue
yoyo	3	2	5

How this table gets defined and build in taxonomy terms will be discussed later. For the moment we focus on the dimensions and their ancestors. As you can see we report on all the members of the colour dimension. What we also see is that we do not have a grand total of toys produced. That is

because we did not leave space in the dimension to hold the grand total. We can accommodate for this by adding an extra member, called `cljpn:allcolours` with label `total`. We add this member as a domain and dimension-default to the dimension. And we will have the colours dimension look like this

```
-  Colors [dimension]
    -  All colors [domain]
        -  Red [member]
        -  Yellow [member]
        -  Blue [member]
```

If we now look at the table we see that there is an extra layer of header labels, and an extra column to hold the grand total.

				All colors
	Red	Yellow	Blue	
yoyo	3	2	5	10

We could go further, we could add a `primary color` member, which would act as a domain for red, yellow and blue, and `other color` to count the newly started silver and gold line.

						All colors
			Primary color		Other color	
	Red	Yellow	Blue	Silver	Gold	
Yoyo	3	2	5	2	2	14

So dimensions add an axis to the data, the members of the dimension is where we report on. In this case, we knew before hand which colors we produce, so the amount of possible members is finite. We can declare every color we produce in our taxonomy and we can report on each one of them. When all members of a dimension are known and defined in the taxonomy we speak of an *explicit dimension*.

But it also happens that the number of possibilities can not be known beforehand. Think of a report where we have to specify the income of each board member. We would report this for instance on a concept `income`. And we would create a `boardMemberDimension` dimension. So far this is identical to the colors dimension. But it is impossible for the taxonomy writer to know all the names of all

the board members within all reporting companies. So the members for this dimension are absent in the taxonomy.

These kind of dimensions are called *typed dimensions*. They do not dictate a finite list of options but instead offer a `type`. For example `string255`. Which means, in this case, that the name of the director must filled in, and must be a string of maximum 255 characters. Typed dimensions are in most cases of a text item type of some sort.

13.1 Dimensions the xml part

Now that we have seen that dimensions can add detail to a report beyond the basic aspects it's time to see how these hierarchies are defined within a taxonomy.

Like the presentation linkbase, dimensions have their own linkbase files. Let's look at the xml definition of the following dimension

```
- Classes of intangible assets
   - Total of intangible assets [domain] (totalLabel)
   - Costs relating to the incorporation and issuance of shares [member]
   - Other intangible assets [member]
```

Let's start with the beginning of the xml needed to create this dimension.

```
<link:linkbase
    xmlns:gpl="http://xbrl.org/2013/preferred-label"
    xmlns:link="http://www.xbrl.org/2003/linkbase"
    xmlns:xbrldt="http://xbrl.org/2005/xbrldt"
    xmlns:xlink="http://www.w3.org/1999/xlink"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://xbrl.org/2013/preferred-label
↪   http://www.xbrl.org/2013/preferred-label.xsd">
  <link:roleRef roleURI="urn:kvk:linkrole:classes-of-intangible-assets"
↪   xlink:href="kvk-linkroles.xsd#kvk-lr_ClassesOfIntangibleAssets"
↪   xlink:type="simple"/>
  <link:arcroleRef
    arcroleURI="http://xbrl.org/int/dim/arcrole/dimension-domain"
    xlink:href="http://www.xbrl.org/2005/xbrldt-2005.xsd#dimension-domain"
    xlink:type="simple"/>
  <link:arcroleRef
    arcroleURI="http://xbrl.org/int/dim/arcrole/domain-member"
    xlink:href="http://www.xbrl.org/2005/xbrldt-2005.xsd#domain-member"
    xlink:type="simple"/>
```

Compared to the presentation linkbase this one needs a lot more namespaces defined. Partly because the `preferred label` role is used. We see a link to a roleRef, just as with the presentation linkbase. This roleRef also must have a `<link:usedOn>link:definitionLink</link:usedOn>` to have a definition linkbase attached. Then we load the definition of the arcroles we are going to use via the `link:arcroleRef`. In this case:

- dimension-domain
- domain-member

Further on we will see that there are even more arcroles for the definition available, but for dimensions, these two suffice.

Now we can start to define our dimension. In this type of linkbase we use the `link:definitionLink`.

```
[continued]
  <link:definitionLink
    xlink:role="urn:kvk:linkrole:classes-of-intangible-assets-small-medium"
    xlink:type="extended">
```

Followed by Arcs and Locators. In this case we use `link:definitionArc`.

```
[continued]
    <link:definitionArc
      gpl:preferredLabel="http://www.xbrl.org/2003/role/totalLabel"
      xbrldt:usable="true"
      xlink:arcrole="http://xbrl.org/int/dim/arcrole/dimension-domain"
      xlink:from="jenv-bw2-dim_ClassesOfIntangibleAssetsAxis_loc"
      xlink:to="jenv-bw2-dm_IntangibleAssetsMember_loc"
      xlink:type="arc"/>
```

First we see that the taxonomy designer wants a total label displayed along with the concept instead of the standard label. This is done by defining a preferred label on the arc, just as in the presentation linkbase.

Next we see a `xbrldt:usable`, I'll leave the explanation for after we covered the whole file.

The arcrole has a value of `dimension-domain` and this is thus the top of this hierarchy, where the `xlink:from` represents the dimension and the `xlink:to` the domain. Exactly like the presentation linkbase where the from-to-relationship represented parent-child arcrole with parents on the left (from) and children on the right (to). In contrast to the presentation network where each relationship is represented by the same arcrole, each subsequent arcrole in the dimension network **must** be `domain-member`. That means that each member in theory can act as the domain of it's children.

The rest of the file consists of two more definition-arcs and 4 locators to link to the concepts that represent the members and give them their labels.

```
[continued]
    <link:definitionArc
      order="1"
      xbrldt:usable="true"
      xlink:arcrole="http://xbrl.org/int/dim/arcrole/domain-member"
      xlink:from="jenv-bw2-dm_IntangibleAssetsMember_loc"
      xlink:to="jenv-bw2-dm_CostsIncorporationShareIssueMember_loc"
      xlink:type="arc"/>
    <link:definitionArc
      order="3"
      xbrldt:usable="true"
      xlink:arcrole="http://xbrl.org/int/dim/arcrole/domain-member"
      xlink:from="jenv-bw2-dm_IntangibleAssetsMember_loc"
      xlink:to="rj-dm_IntangibleAssetsOtherMember_loc"
      xlink:type="arc"/>
    <link:loc
      xlink:href="jenv-bw2-axes.xsd#jenv-bw2-dim_ClassesOfIntangibleAssetsAxis"
      xlink:label="jenv-bw2-dim_ClassesOfIntangibleAssetsAxis_loc"
      xlink:type="locator"/>
    <link:loc

↪   xlink:href="jenv-bw2-domains.xsd#jenv-bw2-dm_CostsIncorporationShareIssueMember"
      xlink:label="jenv-bw2-dm_CostsIncorporationShareIssueMember_loc"
      xlink:type="locator"/>
    <link:loc
      xlink:href="jenv-bw2-domains.xsd#jenv-bw2-dm_IntangibleAssetsMember"
      xlink:label="jenv-bw2-dm_IntangibleAssetsMember_loc"
      xlink:type="locator"/>
    <link:loc
      xlink:href="rj-domains.xsd#rj-dm_IntangibleAssetsOtherMember"
      xlink:label="rj-dm_IntangibleAssetsOtherMember_loc"
      xlink:type="locator"/>
  </link:definitionLink>
</link:linkbase>
```

We've now covered the basics of creating dimensions to add detail to our reports. With this principle we can create as much axis as we want. And with the help of the hypercube we can define which axis/dimensions one must use to create a fact that is valid against the taxonomy.

I owe you however an explanation of `xbrldt:usable` When this has the value of *true*, this means that this member or domain can be used as a value in a context. When `xbrldt:usable` is *false*, the member (or domain) is only used to group the ancestors but can not be used as a context value.

Or to put it in other words, if the `to` relationship has `usable="true"` then the member is a cell heading in a table. On the other hand if `usable="false"` then the member is only there to group the children.

In the second colours example we introduced the domains `primary color` and `other color` and drew a table. When we write out the `color` dimension as a tree and add the usable attribute we see the following

```
-  Colors [dimension]
    -  All colors [domain] usable = true
        -  Primary colors [domain] usable = false
            -  Red [member] usable = true
            -  Yellow [member] usable = true
            -  Blue [member] usable = true
        -  Other colors [domain] usable = false
            -  Gold [member] usable = true
            -  Silver [member] usable = true
```

So in this case neither Primary colors nor Other colors add a column to the table, but only group their children. The all colors domain is usable, so the total has its own column in the table.

I have mentioned that there is also another type of dimension, those whose members can't be known while writing the taxonomy, the so called `typed dimensions`. A typed dimension is constructed by defining the name of the dimension and the type the user provided member must adhere to. So instead of linking domains and members, this dimension type has only one child. This child is always a type definition.

The dimensional abstract in xml

```
<xs:element
    abstract="true"
    id="bzk-wnt-dim_NameOfSeniorOfficialAxis"
    name="NameOfSeniorOfficialAxis"
    nillable="false"
    substitutionGroup="xbrldt:dimensionItem"
    type="xbrli:stringItemType"
    xbrldt:typedDomainRef="bzk-wnt-domains.xsd#bzk-wnt-
↪   dm_NameOfSeniorOfficialTypedMember"
    xbrli:periodType="duration"/>
```

the new attribute `xbrldt:typeDomainRef` points to a concept which holds the type definition. Looking at this element we see:

```
<xs:element
    abstract="false"
    id="bzk-wnt-dm_NameOfSeniorOfficialTypedMember"
    name="NameOfSeniorOfficialTypedMember"
    nillable="false"
    type="nl-types:string255"/>
```

Which means we can fill out anything we want as long as it's a string of 255 characters or less. Typed dimensions can hold lots of different things, like buildings, pieces of land, names of board members, ships, etc.

As we saw earlier, dimensions add aspects to facts, so we can report facts on different detail levels. Dimensions also play a big role in the next resource we will examine; the hypercube.

14 Hypercubes

A well structured hypercube can be seen as a validation table. It binds the reportable items, or line-items as they are sometimes called, to the dimensions that need to accompany the value to be valid. Hypercubes are multi-dimensional tables. On paper we can draw 2 dimensional tables, a cube is of course a 3-dimensional table. Within XBRL tables can have more axis than 3. Although it is possible, with repetition to draw 4 or more axis on paper, it gets complex and unreadable rather soon.

The hypercube is like dimensions part of the `link:definitionLink` and can be defined in it's own linkbase file. Let's first take a look at how we can define a hypercube and link dimensions to it.

```
<link:linkbase
    xmlns:link="http://www.xbrl.org/2003/linkbase"
    xmlns:xbrldt="http://xbrl.org/2005/xbrldt"
    xmlns:xlink="http://www.w3.org/1999/xlink">
  <link:roleRef
      roleURI="urn:kvk:linkrole:balance-sheet-micro-entities"
      xlink:href="kvk-linkroles.xsd#kvk-lr_BalanceSheetMicroEntities"
      xlink:type="simple"/>
  <link:roleRef
      roleURI="urn:kvk:linkrole:basis-of-preparation"
      xlink:href="kvk-linkroles.xsd#kvk-lr_BasisOfPreparation"
      xlink:type="simple"/>
  <link:roleRef
      roleURI="urn:kvk:linkrole:financial-statement-separate"
      xlink:href="kvk-linkroles.xsd#kvk-lr_SeparateFinancialStatement"
      xlink:type="simple"/>
```

We start with defining the used namespaces, and the roleRefs we want to use. This should slowly seem familiar, it is how we build all our linkbase files.

```
[continued]
  <link:arcroleRef
      arcroleURI="http://xbrl.org/int/dim/arcrole/all"
      xlink:href="http://www.xbrl.org/2005/xbrldt-2005.xsd#all"
      xlink:type="simple"/>
  <link:arcroleRef
```

```
      arcroleURI="http://xbrl.org/int/dim/arcrole/hypercube-dimension"
      xlink:href="http://www.xbrl.org/2005/xbrldt-2005.xsd#hypercube-dimension"
      xlink:type="simple"/>
```

Two new arcroles. The first `all` is used to define the hypercube itself. There also exists a `notall` arcrole, also used to define a hypercube, but in a negative sense. It defines not valid concept dimensional combinations. The `notall` hypercubes are not frequently used, and when they are, they are mostly used in combination with an `all` hypercube to exclude some unwanted combinations.

The second arcrole; `hypercube-dimension` has a descriptive name from itself. It binds dimensions to the hypercube. The hypercube will be the `xlink:from` in the arc and the dimension the `xlink:to`

```
[continued]
  <link:definitionLink
      xlink:role="urn:kvk:linkrole:balance-sheet-micro-entities"
      xlink:type="extended">
    <link:definitionArc
        order="3"
        xbrldt:closed="true"
        xbrldt:contextElement="scenario"
        xlink:arcrole="http://xbrl.org/int/dim/arcrole/all"
        xlink:from="sbr-dim_ValidationLineItems_loc"
        xlink:to="sbr-dim_ValidationTable_loc"
        xlink:type="arc"/>
```

We then start again with a `link:definitionLink` like with the dimension, followed by a `link:definitonArc`. Here the attributes of the hypercube are assigned. The first new one is `xbrldt:closed` with here a value of `true`. The value of this attribute alters how the hypercube validates the facts. A `closed` hypercube means that all facts must offer a valid member for each dimension linked to the hypercube, unless there is a dimension-default for a dimension, in which case that dimension can be omitted. When a hypercube on the other hand is `open` it is possible to add members from dimensions not linked to the hypercube, provided that the hypercubes dimensional needs are fulfilled.

Then we see `xbrldt:contextElement`. This is more or less a legacy attribute. It can have two values, either `scenario` or `segment`. Advice is to choose one and stick to that, and if you have no legacy to maintain, `scenario` would be the preferred choice.

The `xlink:arcrole` tells us that this is a `all` hypercube. Followed by the `from` and `to`. Later we will follow the locators to see what this top node of the hypercube really is.

```
[continued]
    <link:definitionArc
        order="1"
        xbrldt:targetRole="urn:kvk:linkrole:basis-of-preparation"
        xlink:arcrole="http://xbrl.org/int/dim/arcrole/hypercube-dimension"
        xlink:from="sbr-dim_ValidationTable_loc"
        xlink:to="jenv-bw2-dim_BasisOfPreparationAxis_loc"
        xlink:type="arc"/>
```

Then we see the next `link:definitonArc`, this time it's a `hypercube-dimension` arc, which tells us that the `from` is a `hypercube` and the `to` must me a dimension.

New is `xbrldt:targetRole` on this arc. It tells us to ignore the roleUri of the `link:definitionLink` but instead look at this role. With this mechanism it is possible to define a dimension once and reuse it on many hypercubes.

With this Arc we've bound the *basis of preparation* axis to the *balance sheet* In the last part, that I will present as a whole, we will attach another axis; *financial statement separate*, which is a dimension with one member; *separate*. After that follow the locators that point to their elements and bind to the *definitionArc*.

```
[continued]
    <link:definitionArc order="2"
        xbrldt:targetRole="urn:kvk:linkrole:financial-statement-separate"
        xlink:arcrole="http://xbrl.org/int/dim/arcrole/hypercube-dimension"
        xlink:from="sbr-dim_ValidationTable_loc"
        xlink:to="jenv-bw2-dim_FinancialStatementsTypeAxis_loc"
        xlink:type="arc"/>
    <link:loc
        xlink:href="jenv-bw2-axes.xsd#jenv-bw2-dim_BasisOfPreparationAxis"
        xlink:label="jenv-bw2-dim_BasisOfPreparationAxis_loc"
        xlink:type="locator"/>
    <link:loc
        xlink:href="jenv-bw2-axes.xsd#jenv-bw2-dim_FinancialStatementsTypeAxis"
        xlink:label="jenv-bw2-dim_FinancialStatementsTypeAxis_loc"
        xlink:type="locator"/>
    <link:loc
        xlink:href="sbr-dimensional-concepts.xsd#sbr-dim_ValidationLineItems"
        xlink:label="sbr-dim_ValidationLineItems_loc"
        xlink:type="locator"/>
    <link:loc
        xlink:href="sbr-dimensional-concepts.xsd#sbr-dim_ValidationTable"
        xlink:label="sbr-dim_ValidationTable_loc"
        xlink:type="locator"/>
```

```
  </link:definitionLink>
</link:linkbase>
```

So this definition Linkbase defined a hypercube and bound that to two extra dimensions. What I left out is the hypercube itself. Let's go back to the top of the file and look at the first definitionArc with some attributes stripped, and their locators.

```
    <link:definitionArc
        xlink:arcrole="http://xbrl.org/int/dim/arcrole/all"
        xlink:from="sbr-dim_ValidationLineItems_loc"
        xlink:to="sbr-dim_ValidationTable_loc" />
    <link:loc
        xlink:href="sbr-dimensional-concepts.xsd#sbr-dim_ValidationLineItems"
        xlink:label="sbr-dim_ValidationLineItems_loc"
        xlink:type="locator"/>
    <link:loc
        xlink:href="sbr-dimensional-concepts.xsd#sbr-dim_ValidationTable"
        xlink:label="sbr-dim_ValidationTable_loc"
        xlink:type="locator"/>
```

There are two locators pointing to concepts. The `to`, which is the hypercube in this case, look like this:

```
<xs:element
  id="sbr-dim_ValidationTable"
  name="ValidationTable"
  abstract="true"
  nillable="true"
  substitutionGroup="xbrldt:hypercubeItem"
  type="xbrli:stringItemType"
  xbrli:periodType="duration"/>
```

The first things we notice is that this is an abstract and that the `substitutionGroup` is not `xbrli:item`, but `xbrldt:hypercubeitem`. We'll get to that. First we take a look at the `from`, which is also an abstract.

```
<xs:element
   id="sbr-dim_ValidationLineItems"
   name="ValidationLineItems"
   abstract="true"
   nillable="true"
```

```
  substitutionGroup="sbr:primaryDomainItem"
  type="xbrli:stringItemType"
  xbrli:periodType="duration"/>
```

As we've seen, abstracts are incapable of holding values, they are just structural nodes and they should have children which are real concepts to hold any value. At the start I explained that hypercubes bind concepts and dimensions together. So far we've seen the dimensions. To find the concepts we must follow the network which is provided by the Abstract `ValidationLineItems`.

Remember that we referenced a linkrole by it's URI `roleURI="urn:kvk:linkrole:balance-sheet-micro-enti` the top of this definition linkbase? And that the linkrole itself acts as a binder for connected hierarchies. When we've read all the linkbases and have all the hierarchies linked to this linkbase we see that we've discovered another definition linkbase, this time the so called **line items** or the concepts we report on.

Let's have a look at this definition as well

```
<link:roleRef
    roleURI="urn:kvk:linkrole:balance-sheet-micro-entities"
    xlink:href="kvk-lr_BalanceSheetMicroEntities"
    xlink:type="simple"/>
<link:arcroleRef
    arcroleURI="http://xbrl.org/int/dim/arcrole/domain-member"
    xlink:href="http://www.xbrl.org/2005/xbrldt-2005.xsd#domain-member"
    xlink:type="simple"/>
<link:definitionLink
    xlink:role="urn:kvk:linkrole:balance-sheet-micro-entities"
    xlink:type="extended">
  <link:definitionArc
      order="1"
      xlink:arcrole="http://xbrl.org/int/dim/arcrole/domain-member"
      xlink:from="sbr-dim_ValidationLineItems_loc"
      xlink:to="jenv-bw2-i_Assets_loc"
      xlink:type="arc"/>
  <link:definitionArc
      order="2"
      xlink:arcrole="http://xbrl.org/int/dim/arcrole/domain-member"
      xlink:from="sbr-dim_ValidationLineItems_loc"
      xlink:to="jenv-bw2-i_AssetsCurrent_loc"
      xlink:type="arc"/>
```

I'll omit the rest of this file, there are more line-items in this balance sheet, what's important is that the roleURI of the roleRef is the same as the definition of the hypercube. What we also can see is that

the line-items hierarchy is nothing less than a dimension network, all the `link:definitonArcs` are of the `domain-member` role.

All this together gives us a hypercube with line-items and two additional dimensions on top of the implicit dimensions of entity, period and sometimes unit. When we draw this hypercube as a table we can represent it like so:

	Fiscal	Commercial
	separate	**separate**
Assets		
Current assets		
Other current assets		
Non-current assets		
Balance sheet before of after appropriation of result		
[continued]		

The labels in the left column are from the Concepts. The *what* we want to report. At the top we see two heading rows, each the result of the dimensions attached to this hypercube. The top row comes from the `basis of preparation` dimension. It has two usable members in it's hierarchy; `Fiscal` and `Commercial`. So both members add a column to the table.

Next we have the second dimension `Financial statements type`. This dimension has (for this report) just one member; `separate`. This member is also usable and repeated for each member of the previous header row. It is possible that that the `Financial statements type` has two members. `Separate` and `Consolidated`. If that is the case, the hypercube heading could look like this:

Fiscal		Commercial	
separate	consolidated	separate	consolidated

With all this extra tools (hypercubes, dimensions, domains and members) provided by the taxonomy we also introduced a way to validate reports against the taxonomy. Dimensions add the possibility to report on an, in theory, unlimited amount of aspects. Hypercubes offer a way to define which of these dimension members can be used to report a value as a fact. Together they offer what we call dimensional validity.

15 Dimensional validity

With the introduction of dimensions and hypercubes there is by definition something that we call dimensional validity. We saw that there two types of hypercubes `all` and `notall`. The first one, the most common denotes a hypercube with *n*-dimensions and *i* concepts as line-items. For a fact to be valid in this hypercube it must pick a *member* from each dimension from that hypercube.

If we look again at this tabular representation of a hypercube from the previous chapter:

	Fiscal	Commercial
	separate	**separate**
Assets		
Current assets		
Other current assets		
Non-current assets		
Balance sheet before of after appropriation of result		
[continued]		

We see 5 line-items, and 2 *explicit dimensions*. When we want to denote *Assets* in an instance we **must** add one of the two members of the *basis of preparation* dimension as an aspect as well as the one member of the *financial statement separate* dimension.

This is true for each line-item we state in the instance. They all must refer to both dimensions.

An instance that would refer a third dimension with the fact *Assets* would trigger an error from an xbrl processor, because of a dimensional mismatch.

If a dimension has a **dimension default** member, which is a *catch-all* member, then it is possible to omit the dimension from the fact. It is also prohibited to explicitly add the *default member* to the fact. Suppose you have a *Geography* dimension, with *Europe*, and the other continents as domains. And Europe has *France*, *Spain*, *Germany* and *Other countries* as its members. Then it could be possible and logical to make *Other countries* the dimension default. But dimension defaults are tricky in the

sense that they also attract other facts, that do omit the dimension, so a parser might try to add the default member to the fact even though it should not be report on that dimension. They can work for small taxonomies with few dimensions, but when taxonomies grow this false reporting on a default member may start to occur.

So far only the `all` hypercube with `closed` is *true* has been covered. It's also possible to have an `all` hypercube which has `closed` set to *false*, this is also referred to as an *open hypercube*. If that was the case with the example balance sheet, we could add a member from any dimension that is known in the taxonomy. And because it is possible to extend a taxonomy with your own dimensions and members one could add granularity not provided and still have a valid instance. But this would only be possible on a *open* hypercube (`closed` =*false*).

The second type of hypercube is the opposite in terms of validation of the `all` hypercube. The `notall` hypercube defines all dimensional combinations that are not allowed in an instance. These hypercubes seldom exist on their own. If you find them in a taxonomy they are usually used to exclude some combinations of dimensions in a *all* hypercube.

16 Table linkbase

One does not escape the table linkbase. It's designed to be a presentation aide where normal hypercubes are too simple, but also a calculation aid via parameters and formulas, as well as a way to get repeating values easily added to the table (think period, entity, unit) by ways of parameters. Lastly you are able to define the width of for example the period. In a taxonomy there is no way to proscribe for which period(s) one is supposed to fill out the report. This is normally dictated by the accompanying filing rules. By using table definitions one is able to proscribe this requirement with the taxonomy.

Because the options to have the outcome of a formula as a value for either a member or a line item and the flexibility of building the visual outcome they are the perfect tool to present complex data in a 2-dimensional space. But the flexibility comes at the cost of complexity. Because so much is possible, on so many layers of the construction, the definition is both hard to write and to read.

Maybe because of this, their usage is restricted to a couple of taxonomies that I know of. But don't let it scare you, a table is also just an hierarchy with a table definition on top. But before we can look at the table definition in a taxonomy we first need to introduce a new term *generic links*.

16.1 Generic links

Instead of adding another link type with predefined semantics and syntax, like the presentation-, definition- or label linkbase, the xbrl organisation added the generic link. With the help of these generic links we can extend our taxonomy with labels for other elements than concepts. And with generic links we can start the definition of an xbrl table. A table is just like all the other parts that provide structure defined in a linkbase file, so it is just another hierarchy. And it starts just like any other linkbase file with a `link:linkbase` tag with the used namespaces as attributes. Then we link to the used roleRef that functions as the top of the table (and possibly also a presentation or definition link as well.)

What then follows is a generic link. A generic link `gen:link`, in combination with a generic arc `gen:arc`, can establish relationships between arbitrary elements. Here the generic link will be used to "bind" the table hierarchy to the roleRef.

I will not give an XML example for the table. Mainly because it would be just 'a' table and it is a lot of XML to digest. A table linkbase requires more namespaces than the ones we introduced before. That is partly because of the generic link itself, and partly because of the possibilities tables offer. If for instance you use formula and filters to select parts of an hierarchy you need to include the formula namespace as well.

Then we link to a roleRef, just like the other linkbase types.

Then we will use a `<gen:link>` generic link which links to the roleRef.

The next part consists of the table definition itself. What you can and can not do within a table is specified in the table specification. The ultra short version leaving tons of details and edge cases out:

- you start with a `<table:table>`, whose children are
- two or more `<table:breakdown>` nodes (x, y and optional z axis)
- whose children are either
  - a `<table:conceptRelationshipNode>`,
  - a `<table:dimesnionRelationshipNode>`,
  - a `<table:aspectNode>`,
  - or a `<table:ruleNode>`

There is a lot more XML-glue involved to get these structured in a hierarchy, but these are to main parts of a table definition.

The table is always a single node. The breakdowns, which define the axis of the table can have breakdowns on the same axis as siblings. So it is possible to have two, three or even more breakdowns which all define a part of the same axis. Under the breakdowns, which by themselves have no relationship with any concept, we could have four different types of children. The *conceptRelationshipNode* and the *dimensionRelationshipNode* are both 'leaves', they point to a hierarchy in the taxonomy and they add their hierarchy to the table as either rows (y-axis) or columns (x-axis). When there is a z-axis defined in the breakdowns, which is usually a dimension, the table should be repeated for each member in that dimension.

The *aspectNode* adds a certain aspect to the table. This can be any of the known aspects like *entity*, *period*, *unit* or a dimension. There can be only one child to an aspectnode, which makes it also a leave of the table definition.

The *ruleNode* on the other hand is more versatile. It can have different types of children. All four types of breakdown children can also be child of a rulenode. So the *conceptRelationshipNode*, the *aspectNode* and the *dimensionRelationshipNode* can be children of a *ruleNode*. A rulenode can also be the child of a rulenode, which can recurse when there is yet another rulenode as child.

The rulenode can also have a `<table:ruleSet>` as a child. This tag can define *formulas* that can add values to context of the facts displayed, found elsewhere in an instance. This way you can help filers so they don't have to repeat some values over and over again.

The use of tables in official taxonomies is not widespread. Because of the complexity to create them in the taxonomy combined with the added value to a average user the benefits are debatable. Tables sure make nice output when you display or print a report. And they do give some help to the filers, but the move to inline xbrl makes, in my opinion, the table as a method of view obsolete.

With that in mind, and knowing this is just a quick guide I recommend you to examine the specifications if you want to dive any deeper into tables. A couple of (online) viewers and tooling are capable to preview tables. Tooling can also help you create a table with the help of a GUI. Understanding the specification is only needed when you want to write your own taxonomy or even more ambitious write your own xbrl parser.

17 Calculations and formula

The last types of hierarchy we will discover are the different calculations- and formula linkbases. In XBRL it is possible to add calculations to add and distract values from one another. There are different implementations which each have their own limitations. ## XBRL Calculations 1.0 The base xbrl 2.1 specification contains the original calculation link, which is referred to as Calculations 1.0. The calculation link is like the others it uses `calculationArc`'s with the role `http://www.xbrl.org/2003/arcrole/summation-item`.And this time i use the whole URI because the other Calculations linkbase 1.1 also uses the arcrole `summation-item`, only with the URI `https://xbrl.org/2023/arcrole/summation-item`. But let's focus on the Calculations 1.0 for now.

To be part of this hierarchy a concept must be of a numeric type. Also must all members of this hierarchy be of the same `period type`. The hierarchy is build up in a way that the children add up to the value of the parent.

Calculations are defined in a taxonomy using summation-item relationships

- The concept identified as the source of the relationship is referred to as the total concept.
- The concept identified as the target of the relationship is referred to as a contributing concept.
- The value of the defining arc's weight attribute is referred to as the contribution weight.

Lets look again at a couple of line items of the hypercube we created in the hypercube chapter

- Assets
  - Current assets
  - Other current assets
  - Non-current assets

We want the three underlying concepts to add up to the to level *Assets*. To accomplish this we create a calculation network that resembles the definition network.

```
<link:linkbase
  xsi:schemaLocation="http://www.xbrl.org/2003/linkbase
↪  http://www.xbrl.org/2003/xbrl-linkbase-2003-12-31.xsd">
  <link:roleRef
```

```
    roleURI="urn:kvk:linkrole:balance-sheet-micro-entities"
    xlink:type="simple"
    xlink:href="kvk-linkroles.xsd#kvk-lr_BalanceSheetMicroEntities"/>
  <link:calculationLink
    xlink:type="extended"
    link:role="urn:kvk:linkrole:balance-sheet-micro-entities">
    <link:loc
      xlink:href="jenv-bw2-data.xsd#jenv-bw2-i_Assets"
      xlink:label="jenv-bw2-i_Assets_loc"
      xlink:type="locator"/>
    <link:loc
      xlink:href="jenv-bw2-data.xsd#jenv-bw2-i_AssetsCurrent"
      xlink:label="jenv-bw2-i_AssetsCurrent_loc"
      xlink:type="locator"/>
    <link:calculationArc
      xlink:arcrole="http://www.xbrl.org/2003/arcrole/summation-item"
      xlink:to="jenv-bw2-i_Assets"
      xlink:from="jenv-bw2-i_AssetsCurrent"
      xlink:type="arc"
      order="1"
      weight="1"/>
    <link:loc
      xlink:href="jenv-bw2-data.xsd#jenv-bw2-i_AssetsNoncurrentOther"
      xlink:label="jenv-bw2-i_AssetsNoncurrentOther_loc"
      xlink:type="locator"/>
    <link:calculationArc
      xlink:arcrole="http://www.xbrl.org/2003/arcrole/summation-item"
      xlink:to="jenv-bw2-i_Assets"
      xlink:from="jenv-bw2-i_AssetsNoncurrentOther"
      xlink:type="arc"
      order="1"
      weight="1"/>
    <link:loc
      xlink:href="jenv-bw2-data.xsd#jenv-bw2-i_AssetsNoncurrent"
      xlink:label="jenv-bw2-i_AssetsNoncurrent_loc"
      xlink:type="locator"/>
    <link:calculationArc
      xlink:arcrole="http://www.xbrl.org/2003/arcrole/summation-item"
      xlink:to="jenv-bw2-i_Assets"
      xlink:from="jenv-bw2-i_AssetsNoncurrent"
      xlink:type="arc"
      order="1"
      weight="1"/>
  </link:calculationLink>
</link:linkbase>
```

All three `link:calculationsArc` s share the same *to* concept: *Assets*. All three *from* concepts have a `weight` of `1`. That means we have to *add* the value to the total. When a `calculationArc` has a weight of `-1` the value needs to be subtracted.

With the help of calculations we can validate values inside an instance. The above calculation tells us that Assets **must** be the total of the three concepts mentioned in to `xlink:from`.

The biggest problem with these calculations is that it is prohibited to mix `period` and `duration` concepts in a `link:calculationLink`. So figures from a balance sheet (typically period) can not be added to figures from profit and loss (typically duration). In fact you can only use concepts which share the same context (dimensions, entity, unit, period)

17.1 Calculations 1.1

Besides the restrictions mentioned that facts need to be of the same context to be in a calculation network. Calculations 1.0 has some unwanted behavior with edge cases like rounding errors or not noticing duplicate facts.. Some of the problems were addressed with the introduction of calculations 1.1 which is not a replacement but an addition to the tools available to taxonomy writers to check the consistency of an instance.

The biggest feature is in my opinion the rounding feature where it's possible to define value interval for a higher level concept, that might be reported with an other precision than the underlying concepts.

All in all, calculations have important, but limited power. Their scope is limited to facts within the same context.

The XBRL organisation is working on the next specification **Calculations 2.0**. So far only the requirements have been written down and my estimate is that it will take a couple of years before this has turned in to a specification.

17.2 Formula 1.0

In contrast to calculations, which should only be used for validation and consistency checks, formula are also able to produce facts based on a calculation on facts that are present in the instance. They are also the Swiss army knife for instance validation and comparison. With formula you can rewrite an instance from one taxonomy to an other. They also deserve their own book. Not a brief description which will never do justice to the possibilities.

That said, they might be the silver bullet for your problem, they are also complex and slow. This is because formulae work on the whole set, meaning both the instance(s) as the used taxonomy must

be in memory. Then the function, which is written in XPath 2.0, is executed and the result is given back. This result can be a True or False on a validation rule, a value for a variable that can be used elsewhere, or a complete new fact complete with context as a result of a calculation.

The specification, which was published in 2009. There also exists an Formula overview where the workings of formula get explained. There is a complete list of different formulae. If this is not enough, you are free to write your own implementation of any filters, assertions or calculations not covered by the existing implementations.

18 Glossary

18.0.0.1 Abstract

A special kind of **concept** which are used for titles and subheadings in a report.

18.0.0.2 Aspect

every different axis that contributes to the value becoming a **fact**

18.0.0.3 Axis

In **xBRL** axis is equivalent to **dimension**.

18.0.0.4 Concept

the *what* the value refers to. The concept is one of the **aspects** that contributes to a fact.

18.0.0.5 Context

A values on a **instance** always refers to the context it belongs to. The context consists of different **aspects**

18.0.0.6 Dimension

When it is needed to report detailed information about a **fact** that is beyond the scope of the standard **aspects** an extra aspect can be added by a dimension. The dimension is also called **axis** and is represented by a tree hierarchy. There are two kinds of dimensions; **typed dimensions** and **explicit dimensions**

18.0.0.7 Domain

a domain is an optional child of a **dimension**. A domain groups **members** under the dimension

18.0.0.8 DTS

Discoverable Taxonomy Set. By means of an **entrypoint** a taxonomy can be read and understood (by the right tooling)

18.0.0.9 Entity

When filing a report this is usually done on behave of a person, a company or some other legal form. In **xBRL** we call this the entity The entity contributes a standard **aspect** to a **fact**.

18.0.0.10 Entrypoint

XML Schema file that is the start of a specific filing within a DTS. The entrypoint can be a complete taxonomy but it usually loads more files from the DTS.

18.0.0.11 Explicit dimension

This form of dimension is called explicit because all possible **member** values are known beforehand and can be found in the **taxonomy**.

18.0.0.12 Fact

A fact consists of a value (ex. 23) and a **context** to which this values refers to.

18.0.0.13 Filing rules

Set of rules that are published by the authority. These documents describe all the conditions that must be met by the instance, which could not be expressed in the taxonomy.

18.0.0.14 Generic links

Generic links are used on all the modern stuff. They have been introduced to accommodate mostly **labels** and **definitions** to structural nodes such as linkroles, but also for instance Enumeration option nodes, .

18.0.0.15 Hypercube

A multi-dimensional table representation of **concepts** that share the same aspects.

18.0.0.16 Instance

Or report or xbrl instance. A filing done on the basis of one or more taxonomies. The instance consists of the name of the used **schema**, **contexts** and **values**.

18.0.0.17 Labels

Human readable label for a Concept. These can come in different **languages** and in different 'roles' like standard label, total label, period start label, ….

18.0.0.18 Language

Text elements can be reported in different languages, and so can labels be. When reporting in different languages, the language adds an **aspect** to the fact.

18.0.0.19 Linkbase

Or RoleType. Top of different hierarchies, links together on a roleURI.

18.0.0.20 Member

The leaf nodes in the dimensional hierarchy. These add the actual **aspect** to the **fact**

18.0.0.21 Namespace

Domain to which an object is confined. A namespace is defined by a URN.

18.0.0.22 Prefix

A shorthand to reference a **namespace**

18.0.0.23 QName

Qualified name of anything. This consists of a **namespace** and local name. The namespace is written by it's **prefix** followed by a colon followed by the local name. For example `xbrli:monetaryItemType`.

18.0.0.24 Report

see **Instance**.

18.0.0.25 Schema

Name we use for an XML Schema file. The **entrypoint** is an XML Schema file and is referred to in the **instance** on the schemaRef node.

18.0.0.26 Taxonomy

Set of xml-schema files and linkbases, all written in xml. The taxonomy is discoverable by an **entrypoint**. A taxonomy can hold different entrypoints. A taxonomy consists of concepts and relational information of those concepts with each other.

18.0.0.27 Unit

When reporting a numeric value it should refer to a certain unit. This can be a valuta like Euro or Dollar, but also a measure like kilometers, persons, or complex divisions like Euro per share.

18.0.0.28 XBRL

extensible Business reporting language.

19 Useful books, websites and tools

Most of this quickguide is written with **the XBRL book - the basics of XBRL** by Ghislain Fourny on my desk[1]. Besides my work at Logius where I learned the basics of XBRL with our focus on taxonomies, this book thought me both insights in the workings of XBRL as well as the basics for the XML needed to write or being able to read or understand the raw XML.

When you need to get to the bottom of things, there is no escaping reading the specifications. And read them twice, and then once more. Compare what you've read with known and validated taxonomies. But you seldom need to go beyond the book of Fourny, only when you need to design your own taxonomy a firm understanding of the inner workings and principles of XBRL is needed.

Of course there is the website of https://xbrl.org and a lot of country specific websites, some like the dutch on https://nl.xbrl.org/. Special mention also for https://xbrl.us which also has a lot of information on xbrl.

To file a report against a given taxonomy, things look much brighter. Once a taxonomy has been published it can be read by different tools. These tools give you various insights in the taxonomy and thus what you can report on. I've mentioned Nessie before, which is a taxonomy viewer which holds the NT (Dutch Taxonomy) I co-authored that at Logius. The official viewer is called Yeti which holds the same taxonomies in a slightly different view. A lot of pre-loaded taxonomies can be found at fractalexperience in yet another type of viewer. For filers, the tools are different, they can display the taxonomy in a searchable manner to ease the tagging of a document with the needed information, because seldom does an (i)xbrl instance stand on it's own. It's a machine readable report of existing human readable reports and documents. So a filer wants to quickly find 'the balance sheet', 'mergers and aquisitions'. The presentation linkbase together with the labels, documentation and references offer a way to have this information at a mouse-click when 'tagging' the annual report.

When you want to look at the taxonomy you design yourself you will need local tools. Those whom design and write the taxonomy(-files) will rely on either Interstage XWand from Fujitsu or XMLSpy from Altova or in house built solutions. Most other vendors offer cloud services where either standard taxonomies are used or the vendor writes or dictates the taxonomy you want to use. I'm not saying it's a bad thing, it's just not something that comes cheap when you want to have both full control of your taxonomy design and full operability on the vendors platform.

[1] https://www.amazon.nl/-/en/Ghislain-Fourny/dp/B0DRZL8C4D

When you want to write your own software to interact with a taxonomy and you don't want to spend any money on the forementioned tools, you end up with Arelle. And when you want a taxonomy viewer that works on your local machine with local files and is free to use, you also end up with Arelle. It's far from perfect, but it's maintained and it does the basics by the rules. The software is also certified by the xBRL organization.

1st edition december 2025

Appie Verschoor - Cooljapan