Transforming XBRL into an OWL Ontology

Including a comprehensive introduction to XML, XBRL, NTP, Ontologies and OWL

Version 1.0
(2007-03-13)

Nick Roos 274503
Arjen Vervoort 272407
Maarten Tijhof 275772
Diderik van Wingerden 282798

Seminar Economics and ICT, FEW1904-06

Erasmus University Rotterdam, Erasmus School of Economics

Contents

I. Introduction

The most important part of this document consists of the authors' attempt to transform an XBRL taxonomy into an OWL Ontology. However, before the transformation process is presented, an elaborate and comprehensive introduction is given on XML, XBRL, NTP, Ontologies and OWL to make the reader familiar with these concepts.

This document is written as the final assignment for the course "Seminar Economics & ICT", topic "XBRL" for the Master Programme "Economics and ICT", Erasmus University Rotterdam. All previous assignment results from this course are included in this document.

The intended readers of this paper include Professor Pijls for grading the assignment, people interested in how XBRL can be transformed into an ontology and people looking for a simple introduction to XBRL and related technologies.

This document is structured as follows: part II contains an introduction to XML, part III contains an introduction to XBRL, then part IV contains a short overview of NTP. This is followed by part V on ontologies, RDF and OWL. Then part VI contains the transformation of XBRL to OWL: a first step towards a useful Business Reporting Ontology. Finally part VII contains the list of used images, part VIII the list of examples and tables and part IX references to sources used for writing this document.

II. XML

1. Introduction

The goal of this chapter is to give the reader a basic understanding of XML with respect to the concepts used by XBRL (Extensible Business Reporting Language), a business reporting-specification based on XML which is becoming the de facto standard for exchanging financial reporting information. After reading this chapter, the reader understands enough of XML to delve into the depths of XBRL.

This chapter is structured as follows: paragraph 2 contains a brief introduction to XML; paragraph 3 contains the basics of stylesheets. Then paragraph 4 is about XML Schema, paragraph 5 shows how the ref-attribute works, paragraph 6 gives an introduction to namespaces. Paragraph 7 is about an XML Schema-specific concept: schemalocation. Then paragraph 8 discusses how to import XML-documents into other XML-documents, followed by paragraph 9 about XLink.

2. XML Introduction

As XML is one of the biggest buzzwords today, next to AJAX which incorporates XML as well, a lot has been written about it, its use and its pro's and con's. First off, let's explain what XML is, and what it stands for.

XML is the abbreviation of "eXtensible Markup Language", and is a markup language much like HTML ("HyperText Markup Language"). In contrary to HTML, XML is specifically intended to carry any data, not to present the data in a different form to the user. XML was designed to describe data where HTML provides the means to present the data using predefined tags.

XML doesn't have any predefined tags, but the user is free to structure an XML-document as he or she seems fit, when certain rather simple rules are adhered to of course. As the tags used in a XML-document can be defined differently with every use, XML can utilize an external source to describe and define the tags being used. This can be done in either of two ways. The first is by the use of a Document Type Definition, or DTD. The second is the use of an XML Schema, which will be elaborated on fully in chapters 3 and 8. DTD is an older standard and is regarded deprecated and replaced by the newer and more powerful XML Schema specification.

A very simple example of XML is the following description of a DVD. As we all know, a DVD has several characteristics such as the name of the film, the price, the director, the number of discs, the year of production and so forth. In HTML this data would probably be presented by a table or a list, in XML it would be something like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<DVD>
  <name>Lock, Stock and Two Smoking Barrels</name>
  <director>Guy Ritchie</director>
  <year>1998</year>
  <…>…</…>
</DVD>
```

Example 1 – Simple DVD description

Looking at the given example, several things come to mind. It all starts with the first line; this is called the XML-declaration. It tells the user the document is written in XML and adhering to the XML 1.0 guidelines. Furthermore, it tells us the charset being used is "UTF-8" which is also known as Unicode. This character set not only holds all Latin characters like the ones being used in this document, but it also holds the Arabic, Cyrillic and most of the Modern-Chinese characters.

The XML-file continues with the term <DVD>. It being contained in brackets means it is a tag, not a value. While it is the first tag we encounter, it is the highest

possible tag, also being referred to as root-tag. There can only be one root in an XML-document. Everything in the root-element is part of the 'instance' DVD. The tags within the DVD are "children" of the DVD and thus part of it; in fact they are the characteristics of the given DVD.

Looking at the child elements of the DVD, the first we stumble upon is <name>. In full:

```
<name>Lock, Stock and Two Smoking Barrels</name>
```

Example 2 – An element in full

Everything between < and > stands for the name of the element, and everything between the tags is the actual value of the tag. Here it says the name of the DVD is "Lock, Stock and Two Smoking Barrels".

There are several rules concerning tags, elements and values. For starters, all elements with a value consist of two tags. Every element has to have the basic <element>Value</element> construction, so it has to have a start-tag and an end-tag around the value. Mind you, empty elements are also possible. However, these elements have an altered notation (syntax), there even are three possibilities for an empty element to be notated, and these are the following:

```
<element></element>
<element />
<element/>
```

Example 3 – Empty elements

To make things a little more complicated the tag <element> can have attributes of its own as well, and each attribute can have its own value. An example of this is the following extension to the DVD-example:

```
<DVD id="1">
  <…>…</…>
</DVD>
```

Example 4 – Extension to the DVD example

This says the DVD with id 1 is provided, the author has chosen to make the id not part of the nested attributes of the DVD itself, but to supply it separately in the DVD-tag. Attributes, like tags, are case sensitive, and all attribute-values have to be enclosed in quotes, either single or double.

Special care is needed when single quotes and double quotes are mixed in use. <DVD id="1"> and <DVD id='1'> are equal and valid, while <DVD id='1"> with the quotes mixed is not. <DVD name="Devil's Advocate"> may seem invalid, but is not as the attribute "name" is properly enclosed in double quotes. The single quote in the value is not a problem in this way.

More in general: the attribute value declaration may not contain more then two (opening and closing) of the same type of quotes. Using the one quote type within the other quote type is allowed.

As child elements and attributes both are the characteristics of the parent element they belong to, they're virtually equivalent. However, there are some disadvantages to using attributes compared to child elements. These are:

- attributes cannot contain multiple values (child elements can)
- attributes are not easily expandable (for future changes)
- attributes cannot describe structures (child elements can)
- attributes are more difficult to manipulate by program code
- attribute values are not easy to test against a Document Type Definition (DTD)

It seems the use of attributes is never better that the use of child elements and in most cases it is like this. Only when the XML is used in addition to HTML, attributes can come in quite handy as selection of tags is far easier attribute based, than element-based. As this is a very specific use of XML, we tend to favour the use of child tags as of the above named disadvantages of attributes.

Every element is case sensitive. This means <DVD> and <dvd> are two different elements altogether. There are some more rules concerning elements. These will be shown in the following examples, where only the begin-tags are displayed.

```
<xmlElement>
<01Element>
<.element>
<element name>
```

Example 5 – Wrong XML tag names

These are all wrong, as:

- tags cannot start with "xml"
- tags cannot start with numbers
- tags cannot start with punctuation
- tags cannot contain spaces

Furthermore, some other data can be embedded in a XML-document, like comments, CDATA and processing instructions. This other data also has to adhere to a strict markup, for example; a comment has to start with <!-- and has to end with -->. Anything but -- can be put in between, and comments can be placed anywhere in an XML document.

Yet another rule is that the tags of the elements have to be nested properly to be part of a well-formed XML-document. When the author intents to nest an element in another element, the tags have to be in the right order.

```
<DVD id=”1”>
  <genres>
     <genreType>Comedy</genreType>
     <genreType>Crime</genreType>
     <genreType>Thriller</genreType>
  </genres>
</DVD>
```

Example 6 – Correct element nesting

The example above is correct, in contrast to the following:

```
<DVD id=”1”>
  <genres>
     <genreType>Comedy</genreType>
     <genreType>Crime</genreType>
     <genreType>Thriller</genres>
  </genreType>
</DVD>
```

Example 7 – Incorrect element nesting

Both examples try to list the different genres, with nested genre-tags, but the second time the author gets it wrong. The </genres> tag is placed before the </genreType> tag, so this XML would fail to validate due to it not being well-formed XML.

3. Stylesheets

Considering the fact that an XML-document contains only data, some form of presentation had to be developed. The way of presentation or even transformation is called a stylesheet, and is defined by an eXtensible Stylesheet Language (XSL).
As XSL is a little broad, it was set up to be developed in three parts, including XSL Transformations (XSLT), XSL Formatting Objects (XSL-FO) and XML Path Language (XPath).

XSLT is a way to transform an XML-document to another format; mostly XHTML, HTML, plain text, or an intermediate XML-document to be formatted into a PDF-document later on.

To transform a document from XML into a completely different format XSL-FO has been developed. An XSL-FO document is not like a PDF or a PostScript document. It does not fully describe the layout of the text on various pages. Instead, it describes what the pages look like and where the various content goes. From there, a FO processor determines how to position the text within the boundaries described by the FO document. Finally, XPath provides a way to access different parts of an XML-document.

A stylesheet gives one the ability to completely alter the way an XML-document is presented. XSL is to XML what CSS (Cascading Style Sheets) is to HTML: it can present the data in an altered fashion. As for the presentation of the data, this comparison holds, except that XSL actually outputs the XML into a different format, whereas CSS leaves the HTML intact and the browser parses it for display.

Stylesheets are often being used in a web-based fashion. The XML transformation can be done in two different places, either at the server (server side), or by the computer of the visitor of a webpage (client side).

As XSLT is supported by all modern browsers, this is also the most commonly used part of the XSL languages.

4. XML Schema

4.1. Introduction

As mentioned in the XML introduction, XML has only one purpose and that is the transfer of highly structured data. As XML enables the user to choose their own tags, attributes names and data structures, some form of specification is needed to make the data in an XML-document understandable by all parties.

As XML is just plain text in a special markup, the data is always readable but hardly understandable. Most of the time the author of an XML-document tries to use quite easy to understand element names, but when this is not the case, or when the names are in another language, some explanation might come in handy. This is where XML Schemas come in: they contain the specifications of the elements, attributes and structure of an XML-document.

As previously mentioned, defining the structure of an XML-document can be done by either a Document Type Definition (DTD) or by an XML Schema. The main difference between a DTD and an XML Schema is the fact that the DTD is not in XML, while the XML Schema is an XML-document on its own. The language an XML Schema is written in is called XML Schema Definition. (XSD)

The purpose of an XML Schema is to define the valid building blocks of an XML document, just like a DTD. An XML Schema:

- defines elements that can appear in a document
- defines attributes that can appear in a document
- defines which elements are child elements
- defines the order of child elements
- defines the number of child elements
- defines whether an element is empty or can include text
- defines data types for elements and attributes
- defines default and fixed values for elements and attributes

In contrast to a DTD, an XML Schema also has the following features:

- XML Schemas are extensible to future additions
- XML Schemas are richer and more powerful than DTDs
- XML Schemas are written in XML
- XML Schemas support data types
- XML Schemas support namespaces

When we look at the previously mentioned simple example about the DVDs and extend it a little, we can create both a DTD and an XML Schema for it:

```
<?xml version="1.0" encoding="UTF-8"?>
<DVD>
  <name>Lock, Stock and Two Smoking Barrels</name>
  <director>Guy Ritchie</director>
  <year>1998</year>
  <genre>Comedy</genre>
  <genre>Crime</genre>
  <genre>Thriller</genre>
</DVD>
```

Example 8 – Extended DVD

```
<!ELEMENT DVD (name, director, year, genre+)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT director (#PCDATA)>
<!ELEMENT year (#PCDATA)>
<!ELEMENT genre (#PCDATA)>
```

Example 9 – Part of a DTD

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://www.some-url.com"
    xmlns="http://www.some-url.com"
    elementFormDefault="qualified">

<xs:element name="DVD">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="name" type="xs:string"/>
            <xs:element name="director" type="xs:string"/>
            <xs:element name="year" type="xs:string"/>
            <xs:element name="genre" type="xs:string" maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
</xs:schema>
```

Example 10 – XML Schema (XSD)

As it is quite clear an XML Schema is far less compact compared to a DTD, this is also the (hidden) power of an XML Schema; it provides much more capabilities to the user. These capabilities will be discussed in the following paragraph.

4.2. Element, attribute, simpleType, complexType, sequence, choice

We've described earlier it is possible to add a so-called attribute with its own value to an element. This attribute can give the reader of the XML document extra information of the stored data between the element tags.

Attributes are often used for information that is not part of the data. This can for example be used when the data between the tags is not a name or other textual information but a file. This is unimportant for the data but important for reader programs, attributes can be used to give ID-values to certain elements.

In XML Schema simpleType-elements can be used to give constraints to values: it can be used to give an upper and lower limit to a numeric value. In the following example the element "region" gives the DVD Region Code of the DVD with a lower limit of 0 and an upper limit of 8.

```
<xs:element name="region">
    <xs:simpleType>
        <xs:restriction base="xs:integer">
            <xs:minInclusive value="0"/>
            <xs:maxInclusive value="8"/>
        </xs:restriction>
    </xs:simpleType>
</xs:element>
```

Example 11 – simpleType

It is also possible to have an element that can contain elements. This is different from the previously mentioned simpleType which can only have data in it. For an element which could contain other elements the complexType can be used. In our previous DVD example the DVD-element can have child-elements for the information of a DVD.

```
<xs:element name="DVD">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="title" type="xs:string"/>
            <xs:element name="director" type="xs:string"/>
            <xs:element name="actor" type="xs:string"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
```

Example 12 – complexType

As you can see the element DVD contains a sequence of elements. An element within this complexType can again have children with the use of a sequencelist. A sequence gives an ordered sequence of sub-elements of the topelement, in this case: DVD. Besides the sequence element to create in the XML Schema a choice between elements to be used.

```
<xs:group name="director">
  <xs:choice>
     <xs:element name="name" type="xs:string"/>
        <xs:sequence>
           <xs:element name="firstName" type="xs:string"/>
           <xs:element name="middleName" type="xs:string" minOccurs="0"/>
           <xs:element name="lastName" type="xs:string"/>
        </xs:sequence>
     </xs:choice>
</xs:group>
```

Example 13 – Sequence

With the use of these concepts it is possible to create a complete XML Schema Document (or XSD). For the DVD XML document the XML Schema will be something like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="DVD">
     <xs:complexType>
        <xs:sequence>
           <xs:element name="year" type="xs:string"/>
           <xs:element name="region">
              <xs:simpleType>
                 <xs:restriction base="xs:integer">
                    <xs:minInclusive value="0"/>
                    <xs:maxInclusive value="8"/>
                 </xs:restriction>
              </xs:simpleType>
           </xs:element>
           <xs:element name="name" type="xs:string"/>
           <xs:element name="genres">
              <xs:complexType>
                 <xs:sequence>
                    <xs:element name="genreType">
                       <xs:simpleType>
                          <xs:restriction base="xs:string"/>
                       </xs:simpleType>
                    </xs:element>
                 </xs:sequence>
              </xs:complexType>
           </xs:element>
           <xs:element name="director">
              <xs:complexType>
                 <xs:sequence>
                    <xs:element name="firstName" type="xs:string"/>
                    <xs:element name="middleName" type="xs:string" minOccurs="0"/>
                    <xs:element name="lastName" type="xs:string"/>
                 </xs:sequence>
              </xs:complexType>
           </xs:element>
        </xs:sequence>
     </xs:complexType>
  </xs:element>
</xs:schema>
```

Example 14 – The completed XSD

4.3. Restrictions

Within XML schema, any value would be valid when creating a new instance that follows the schema. These values can be restricted in multiple ways to ensure that only valid and correct data is entered and accepted by the processor. These restrictions on the XML elements are called facets.

As said, there are multiple ways to restrict the elements. The most important ones are restrictions on values, restrictions on a set of values and series of values. You can also place restrictions on the use of whitespace characters or on the length of the value within an element.
When using a restriction on values you can ensure that no value below or above a certain value is used. This can be done with the following code, this restricts the element grade to an integer with a value between 1 and 10.

```
<xs:element name="grade">
  <xs:simpleType>
     <xs:restriction base="xs:integer">
        <xs:minInclusive value="1"/>
        <xs:maxInclusive value="10"/>
     </xs:restriction>
  </xs:simpleType>
</xs:element>
```

Example 15 – Value restriction

In a similar way, you can also restrict a set of values. If you have a limited list of values that are possible, you can add these in the restriction. The element classes can only have one of the three values given in the code below.

```
<xs:element name="classes">
  <xs:simpleType>
     <xs:restriction base="xs:string">
        <xs:enumeration value="Seminar I&E"/>
        <xs:enumeration value="ICT & Economics”/>
        <xs:enumeration value="Management Control and ICT"/>
     </xs:restriction>
  </xs:simpleType>
</xs:element>
```

Example 16 – Value set restriction

Besides limiting the real content it is also possible to restrict a series of value. This is especially useful to restrict values to specific needs, for example only letters in lower case of upper case. You can also restrict the number of characters. In the example, the shortcode for something like a class or subject, must consist of 3 letters. The first character must be a letter in upper case, the second can be in lower or upper case whilst the last character must be a digit between 0 and 9.

```
<xs:element name="shortcode">
  <xs:simpleType>
     <xs:restriction base="xs:string">
        <xs:pattern value="[A-Z][a-zA-Z][0-9]"/>
     </xs:restriction>
  </xs:simpleType>
</xs:element>
```

Example 17 – Value series restriction

As you can see, restrictions are character based an they can be mixed in any way that fits your specific case. If do not need to restrict every single character in the same way if they all have the same restriction. This can be done as in the following example.

```
<xs:element name="username">
  <xs:simpleType>
     <xs:restriction base="xs:string">
        <xs:pattern value="[a-zA-Z0-9]{6}"/>
     </xs:restriction>
  </xs:simpleType>
</xs:element>
```

Example 18 – Number of characters restriction

In this example, when asking for a username the only values accepted are values that are exactly 6 characters long as restricted by the {6} part in the code. These characters can be either lower or uppercase or a digit between 0 and 9. If you would not want digits in the username, the value of the restriction would be <xs:pattern value="[a-zA-Z]{6}"/>.

The last useful restriction is the restriction on the length of a value that is more flexible then the previous one that limits the length to a fixed value. You could also restrict the length to fit between two given values, for example between 6 and 10 characters long.

```
<xs:element name="username">
  <xs:simpleType>
     <xs:restriction base="xs:string">
        <xs:minLength value="6"/>
        <xs:maxLength value="10"/>
     </xs:restriction>
  </xs:simpleType>
</xs:element>
```

Example 19 – Length restriction

As you see, this only restricts the length of the value and not the actual content. You can also combine these restrictions to create a specific element that exactly fits your needs. The pattern value and length restrictions combined provide a good basis to start with when creating restrictions on the actual data contained in an instance.

4.4. Substitution

With XML Schemas, one element can substitute the other. In practice this means two elements can have the same meaning while being completely different. As long as they are member of the same substitution group, they can be interchanged with each other.

We will show this behaviour with an example of a multi-language login-system, in which the elements are translated into two languages for easy modification by both a Dutch and an English speaking person. We've taken English as the main language, and Dutch as a secondary language.

```
<xs:element name="name" type="xs:string"/>
<xs:element name="naam" substitutionGroup="name"/>
```

Example 20 – A substitution group

The example above shows that the 'name' element is the head element, and the 'naam' element is substitutable for 'name'.

```
<xs:element name="name" type="xs:string"/>
<xs:element name="naam" substitutionGroup="name"/>

<xs:element name="nickname" type="xs:string"/>
<xs:element name="schermnaam" substitutionGroup="nickname"/>

<xs:complexType name="userinfo">
  <xs:sequence>
     <xs:element ref="name"/>
     <xs:element ref="nickname"/>
  </xs:sequence>
</xs:complexType>

<xs:element name="user" type="userinfo"/>
<xs:element name="gebruiker" substitutionGroup="user"/>
```

Example 21 – An XSD snippet

The above schema holds the substitution group mentioned earlier, but it also holds a second substitution group 'nickname'.

According to the schema, a valid XML-document would be like this:

```
<user>
  <name>John Doe</name>
  <nickname>J_D</nickname>
</user>
```

Example 22 – XML document 1

But, because of the defined substitution groups, the document below would also be valid. An element of type 'userinfo' can either be with the name 'user' or 'gebruiker', and an element with the name 'nickname' can be interchanged with an element called 'schermnaam'.

```
<gebruiker>
  <naam>John Doe</naam>
  <schermnaam>J_D</schermnaam>
</gebruiker>
```

Example 23 – XML document 2

Substitution groups can be very powerful tools, and the power might need to be harnessed in order to be kept in control. This can be done by adding the block-attribute to the head element.
Adding this attribute, example 19 would then be like this:

```
<xs:element name="name" type="xs:string" block="substitution"/>
<xs:element name="naam" substitutionGroup="name"/>

<xs:element name="nickname" type="xs:string" block="substitution"/>
<xs:element name="schermnaam" substitutionGroup="nickname"/>

<xs:complexType name="userinfo">
  <xs:sequence>
     <xs:element ref="name"/>
     <xs:element ref="nickname"/>
  </xs:sequence>
</xs:complexType>

<xs:element name="user" type="userinfo" block="substitution"/>
<xs:element name="gebruiker" substitutionGroup="user"/>
```

Example 24 – Example 19 revised

The simple consequence of this of course is that example 20 is still a valid XML-document, while example 21 is not anymore.

A final note on substitution groups is the following: an element is only substitutable with another element if the 'child-element' is of the same type, or a derivative of that type as the 'head-element'.

While this part intends to list the basics of XML to understand XBRL, we know XBRL uses a different technique for internationalisation to the one being used in the examples shown in this chapter. We've chosen this example as it is quite easy to understand, internationalisation in the DVD-example also was an option but it wouldn't be intuitive.

4.5. Nullable and Nillable

It appears that here are a lot of different approaches to the meaning of nullable and nillable. Some say that when something is nullable it may have a value of null but must be present. Others say that a nullable element should not be always present. The same goes for nillable. There is no information available that specifies the exact use of both attributes so we present our vision on these two attributes.

Nullable should be used when an element is not required in an instance of a schema, its usage is allowed but not required. When a user decides not to use this element it can be left out.

Nillable is different, al be it delicate. When an element has the nillable attribute it must be present in the instance document. However, the user has the option to leave its value empty.

The main difference is that in the final instance document is that other people viewing the document can see that some elements are not filled in when using nillable. If nullable would be used, the viewers would not have knowledge of the nullable element.

5. The Ref-attribute

The ref-attribute in an XML Schema is used as a reference pointer. First, you must define an element or attribute in the document. At this point you define the name and content. After that you can duplicate this element or attribute by using the ref attribute with just the name defined in the previous step.

For example, at the start of the XML Schema you could define the following element:

```
<element name=”title” type=”string”/>
```

Further on in the document you can reference to this element by using the following statement:

```
<element ref=”title”/>
```

6. Namespaces

When using multiple XML files you are likely to run into naming problems. This is because XML has no predefined element names. When you use the same name in two different documents and they are used together you will get an element name conflict. You can avoid this by using namespaces, this allows you to put a prefix before the element names which associates all child elements with the same namespace. Because of this, it is possible to tell the two elements apart.

File 1 contains the following lines of code:

```
<DVD>
    <title>Lock, Stock and Two Smoking Barrels</title>
    <director> Guy Ritchie</director>
</DVD>
```

Example 25 – File 1

File 2 contains different code but uses the same element name:

```
<DVD>
    <title>Lock, Stock and Two Smoking Barrels</title>
    <year>1998</year>
</DVD>
```

Example 26 – File 2

When combining these files containing the code mentioned above will not work. The solution is the XML namespace. This is done by defining a namespace in the start tag of the element. In this definition you link to a specific namespace Uniform Resource Identifier (URI). The only purpose of this is to give the namespace a unique name, it cannot be used anywhere else. It is common practice to use an existing webpage containing information about the namespace although this is not required. You can also define a default namespace, if you do this you don't have to use the prefix on all the child elements. We will show an example that applies both methods. These examples continue with the previous example with File 1 and File 2.

File 1 with prefix “a” from namespace http://www.some-url.com/a:

```
<a:DVD xmlns:a=”http://www.some-url.com/a”>
    <a:title>Lock, Stock and Two Smoking Barrels</title>
    <a:director> Guy Ritchie</director>
</a:DVD>
```

Example 27 – File 1 with prefix namespace

File 2 with a default namespace http://www.some-other-url.com/b:

```
<DVD xmlns= “http://www.some-other-url.com/b”>
    <title>Lock, Stock and Two Smoking Barrels</title>
    <year>1998</year>
</DVD>
```

Example 28 – File 2 with default namespace

The DVD element of the first file is now different then the DVD element found in the second file. You can use both methods to achieve this, however the default namespace gives a cleaner looking code and is easier to use.

7. Schemalocation

XML documents are processed for display on the web, reports in PDF format, etcetera. The creator of the XML document can provide hints for the processor regarding the location of the schema documents. This basically says to the processor that the current XML document conforms to the XML schema mentioned by the schemaLocation attribute. This attribute consists of two values: the namespace name and the schema location. These two values are separated with a blank space.

Before you can use the schemaLocation attribute you have to declare the w3.org namespace because the schemaLocation itself is located in this namespace. Then you have to map it to a prefix, usually xsi because almost everybody uses this prefix.

An example:

```
<dvd xmlns=”http://www.some-url.com/a” xmlsn:xsi=”http://www.w3.org/2001/XMLSchema-
instance” xsi:schemaLocation=”http://www.some-url.com/a a.xsd”>
    <title>Lock, Stock and Two Smoking Barrels</title>
    <director> Guy Ritchie</director>
</dvd>
```

Example 29 – Schemalocation

In this example we tell the processor that the declarations for attributes and elements can be found in the “http://www.some-url.com/a” namespace within the file “a.xsd”. Note that both values are within the same quotation marks.

8. Import

When you are creating an extensive XML document, chances are that the overview of the document gets lost because of the sheer size of it. It is possible to deal with this, you can break up the file into separate, smaller files. These files can then be imported into the current document. These imported files don't have to be in the same namespace as the one you are importing them to. An added bonus, be sided higher maintainability, is that you can re-use the smaller files in other documents.

The syntax of the import element is the following:

```
<import
    id = ID
    namespace = anyURI
    schemaLocation = anyURI
    {any attributes with non-schema Namespace} ..>
    Content: (annotation?)
</import>
```

Example 30 – The import-syntax

The id of the element must be of type ID and be unique within the document. The namespace of the schema and the schemaLocation can also be declared. These three items are optional, they are not required. The main advantage of the import element is that the imported documents can be from different namespaces whereas the include element adds the components of another document that has the same target namespace.

A small example of importing a document into another. Yet again the dvd example, File 1.

```
<dvd xmlns="http://www.some-url.com/a">
    <import href="moreInformation.xml"/>
    <title>Lock, Stock and Two Smoking Barrels</title>
    <director> Guy Ritchie</director>
</person>
```

Example 31 – Importing a document into another

The moreInformation.xml file contains the following:

```
<year>1998</year>
<runTime>107</runTime>
```

Example 32 – A snippet from moreInformation.xml

If you put File 1 through a processor it would treat it as if the file looked like this:

```
<dvd xmlns=”http://www.some-url.com/a”>
    <year>1998</year>
    <runTime>107</runTime>
    <title>Lock, Stock and Two Smoking Barrels</title>
    <director> Guy Ritchie</director>
</dvd>
```

Example 33 – Import result when processed

9. XLink

The XML Linking Language (XLink) specification defines a way to include links to and from resources in XML-documents. The specification is W3C-recommended and currently in its 1.0 version. XLink is comparable to the a-element for hyper linking in HTML, but is much more powerful than that. For one, it provides bi-directional linking, as opposed to the unidirectional linking of the a-element. Furthermore, when combined with other specifications like XPointer and XPath, it provides a very dynamic way of linking XML-documents, but also XML-documents with other resources.

This chapter gives an introduction of the XLink specification. Special attention is paid to the following XLink-concepts: type, from/to, href, arcrole and locator. Simple examples are provided.

9.1. Type

Any XML-element in an XML-document can be used as a link, so the creator is free to use any valid (or well-formed) element name that seems appropriate. By adding special XLink attribute-names, values and sub-elements an XML-element is transformed into an XLink.

XLinks can take different forms, or types, which result in different kinds of links, with different behaviours. The type-attribute specifies the kind of XLink and determines which other attributes and sub-elements can be specified for that specific XLink.
The following types are available:

- simple: simple link
- extended: an extended, possibly multi-resource, link
- locator: a pointer to an external resource
- resource: an internal resource
- arc: a traversal rule between resources
- title: a descriptive title for another linking element

9.1.1. Simple-type

The easiest way of using an XLink is that of the ‘simple’ type. The simple-type looks the most like the a-element available in HTML. Below an example is given for the simple-type XLink:

```
<?xml version="1.0" encoding="UTF-8"?>
<dvd xmlns:xlink=http://www.w3.org/1999/xlink
  xlink:type="simple"
  xlink:href="http://www.imdb.com/tt1254859">
  <name>Lock, Stock and Two Smoking Barrels</name>
  <director>Guy Ritchie</director>
  <year>1998</year>
</dvd>
```

Example 34 - XLink simple-type

Notice the namespace-declaration which is needed for XLink (see also chapter 6).

9.1.2. Extended-type

With the extended-type it is possible to link between multiple resources and also distinguish between the type and direction of linking. The following child elements can be used:

- locator: for a reference to an external resource
- arc: to identify the relationship between resources
- resource: an internal resource

Please look at the following example:

```
<?xml version="1.0" encoding="UTF-8"?>
<hollywood xmlns:xlink="http://www.w3.org/1999/xlink" xlink:type="extended">
    <actor xlink:type="locator" xlink:label="actor" xlink:href="brucewillis.xml"
xlink:title="Bruce Willis"/>
    <movie xlink:type="locator" xlink:label="portfolio" xlink:href="diehard.xml"/>
    <movie xlink:type="locator" xlink:label="portfolio" xlink:href="hostage.xml"/>
    <type xlink:type="locator" xlink:label="genre" xlink:href="action.xml"/>
    <type xlink:type="locator" xlink:label="genre" xlink:href="drama.xml"/>
    <bind xlink:type="arc" xlink:from="actor" xlink:to="portfolio" title="Performed in
movie"/>
    <bind xlink:type="arc" xlink:from="actor" xlink:to="genre" title="Did genre"/>
</hollywood>
```

Example 35 - XLink extended-type

The external resource Bruce Willis (note the descriptionary title-attribute which is used for semantics) is linked to the external resources for two movies (identified as portfolio) and two genres. The arc-type attribute facilitates the linking of resources, providing direction of the links and also a possibility for meaningful description (the title-attribute).

9.2. Arcrole and semantics

So far we have seen the following XLink-attributes with our XLink-examples: type (simple, extended, locator and arc), href, label, from, to and title.
One more complex, yet important, attribute will be discussed here: the arcrole-attribute. This attribute, together with the role-attribute and title-attribute, constitute the semantic attributes of the XLink-specification. In other words: with these attributes some human-understandable meaning can be given.

The title-attribute, as shown before, can be used to give a descriptive title, intended for human interpretation. The title-attribute can be used with the simple, extended, locator, arc and resource type XLinks and is fairly simple by nature. The role-attribute and arcrole-attribute are complementary, as they indicate what role a linked resource or the link itself plays in the context provided. The role-attribute can be used with the simple, extended, locator and resource type XLinks. The arcrole-attribute is used with the arc type-attribute and is used to provide meaning for the relationship between the resources. Since the simple-type is shorthand for the extended-type, arcrole can also be used with the simple type-attribute.

However, there is a big difference: where the title-attribute and role-attribute values are just descriptions in plain text (a string), the arcrole-attribute is actually a URI pointing to a resource containing information about the nature of the relationship.

Below the previous extended-type example is complemented with semantics-attributes:

```
<?xml version="1.0" encoding="UTF-8"?>
<hollywood xmlns:xlink="http://www.w3.org/1999/xlink" xlink:type="extended"
xlink:title="Resources on acting">
  <actor xlink:type="locator" xlink:label="actor" xlink:href="brucewillis.xml"
xlink:title="Bruce Willis" xlink:role="actor"/>
  <movie xlink:type="locator" xlink:label="portfolio" xlink:href="diehard.xml"
xlink:role="movie"/>
  <movie xlink:type="locator" xlink:label="portfolio" xlink:href="hostage.xml"
xlink:role="movie"/>
  <type xlink:type="locator" xlink:label="genre" xlink:href="action.xml"
xlink:role="genre"/>
  <type xlink:type="locator" xlink:label="genre" xlink:href="drama.xml"
xlink:role="genre"/>
  <bind xlink:type="arc" xlink:from="actor" xlink:to="portfolio"
xlink:title="Performed in movie" xlink:arcrole="http://www.imdb.com/acting"/>
  <bind xlink:type="arc" xlink:from="actor" xlink:to="genre" title="Did genre"
xlink:arcrole="http://www.imdb.com/actingstyles"/>
</hollywood>
```

Example 36 - XLink extended-type with semantics

Notice that some labels and roles have the same value: this is because in this example we make no distinction between the actor and genres. In practice it may well be that these values are different. For example: a label with value ‘genre’ can also be indicated as the genre-name (for example: ‘action’) to provide a finer grained way of describing linked resources. This then translates also to a more differentiated connection in the arc-type XLinks: instead of linking actor to all genres, it can now be linked to individual genres.

Also notice that the attribute-values for the arcroles are not existing resources: this is done for the simplicity of this example. In practice they should be: for example when linking to a Cascading Style Sheet, an arcrole could be made to the CSS 2 specification URL.

9.3. Overview tables of available XLink-attributes

It goes beyond this introductionary chapter to address all aspects of the XLink-specification. But to give an idea of these aspects, this paragraph contains two overview-tables of the available XLink-attributes and how these relate to each other. These can be used for/in further studying of the subject.

The following table contains an overview of which type-attribute values (columns) can be used in combination with which global attributes (rows). The R indicates a required-attribute; the O an optional attribute. A blank cell indicates an invalid combination.

	Simple	Extended	Locator	Arc	Resource	Title
Type	R	R	R	R	R	R
Href	O		R			
Role	O	O	O		O	
Arcrole	O			O		
Title	O	O	O	O	O	
Show	O			O		
Actuate	O			O		
Label			O		O	
From				O		
To				O		

Table 1 – combinations of XLink-types and attributes (source: W3C 2001)

The following table contains an overview of the child-types which can be used with a specific parent-type XLink. A blank cell indicates that no child XLink-type can be used. Note that other XML-elements and attributes can be used, but that they constitute no meaning when the XLinks are interpreted.

Parent type	Significant child types
Simple	
Extended	locator, arc, resource, title
Locator	title
Arc	title
Resource	
Title	

Table 2 – XLink-types and significant child types (source: W3C 2001)

III. XBRL

1. Introduction

While our economies and the world keeps on expanding, so does the complexity of the entire system and does the way we communicate. By this we mean the personal communication, but also the business-to-business communication regarding the business' financial data with, for example, the tax authorities.

With this last communication in mind, some problems arise. How to get this financial data from your own company to the tax authority? Until recently this was all done by hand. An employee of the company completed, often in cooperation with the accountant, the tax forms and sent them through the mail to the tax authority. After the mail was received, the letters had to be opened and all be processed by hand. A tax officer read the forms and fed the information into the system manually after which the final taxes were established.
Nowadays, this process is (partly) digitized, as the tax forms are often sent digitally via the internet to the tax authority which speeds the process up significantly.

In our ever becoming more global society standardization of communication is the basis of proper communication. If something is misunderstood because of geographical issues or a personal other view on the subject, the consequences can differ from 'not a problem' to 'major issue'. Coming back to the communication between the tax authority and a company, all data has to be interpreted right. If anything goes wrong this could be catastrophic for both parties. The authority could loose out on income, or the firm has to pay (way) too much taxes.

This is where XBRL comes in. It stands for eXtensible Business Reporting Language, and supplies the user with a comprehensible and agreed upon set of standardised variables, values and concepts. As not all companies are the same it is virtually impossible to catch all used concepts and physical quantities; therefore this has been taken into consideration during the design of XBRL, and is the specification based upon XML. To be precise, XBRL is an extension of XML. To further specify the aspects of XML lies beyond the scope of this chapter, we would like to refer back to chapter 2.

After recognizing the power of XBRL to communicate in a uniform way where all concepts and names are understood in the same manner, quite a few questions come to mind. Is a uniform specification of communication something we really want? Are there any disadvantages to XBRL? What impact does XBRL have on a firm? These questions are all very valid ones, and some are still unanswered. As we do not intend to test XBRL on its capabilities, getting answers to the above posed questions is not part of this guide. We would like to refer to the known resources on the internet and in the literature.

To shortly address the impact XBRL has on a regular firm, we would like to point out the fact that uniformity in communication also has several implications on how and what to communicate. General ledgers and year reports as made by firms differ greatly in lay-out and methodology. For example, firm A likes to calculate its revenues per country while firm B does it per establishment. To communicate in an uniform way, the data to communicate must also be uniform so XBRL handles this by a specification of how to report. This will also be addressed in chapter 6. As the general ledgers have to be the same throughout all the adoptees of XBRL, all companies have to adhere to the rules set in the specification which is quite technical and something completely different to what most company-owners are used to, this puts quite a lot of tress on them to get it done the right way and in time. Again, to continue on this matter lies beyond this chapter's scope.

The XBRL concept is specified in the XBRL-specification created by the World Wide Web Consortium (W3C). The most recent specification of XBRL is XBRL 2.1, issued in 2003, and amended December 2006. This specification only tells the user the way to put the right structure into the XBRL-documents, not the contents or meaning of the data. The latter is described in a taxonomy, described in this chapter.

As XBRL is in fact just plain text with some very peculiar mark-up, it is nearly impossible to get a good view of what is in the document. Being an extension of XML the means to view the document in a comprehensible way are there and a program to do just that is called a viewer. Some companies have created XBRL-viewers, including Semansys who kindly put their ‘Taxonomy Viewer’ on the internet for free.

2. Taxonomy

Being the most important concept of XBRL, taxonomies can be most accurately described as dictionaries or vocabularies. A taxonomy comprises of several parts describing various definitions in the financial world. These include time, all known currencies, debit, credit, debt, projects, and so on.
In short, a taxonomy is the semantics and only contains the meta-data of the financial data which is contained in an XBRL-document.

Typically, a taxonomy consists of six files, all depicted in the figure shown below.

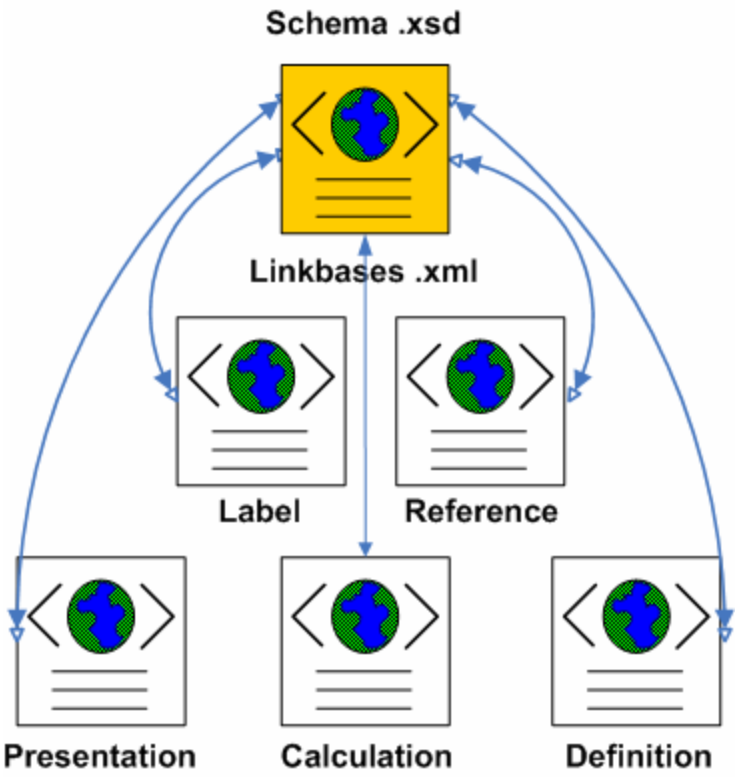


Figure 1 – Parts of a taxonomy

As can be seen from the names of the different files in a taxonomy, it not only consists of the simple declarations but it also provides the means to present the XBRL data in a readable fashion, it determines the calculation to be used and it relates all definitions in the schema.
These files will be elaborated on in the following chapter.

While XBRL is meant to be an international standard, several countries have their own taxonomy, the Dutch version is called 'Nederlands Taxonomie Project' (NTP) or Dutch Taxonomy Project. Creating your own taxonomy is not solely for countries, but it is also being done by specific industries and even some companies.

3. Linkbase

In a linkbase a definition of specific information is given about the elements that are part of the instance document. There are five linkbases existing in every taxonomy in XBRL.

- Reference linkbase
- Labels linkbase
- Presentation linkbase
- Calculation linkbase
- Definition linkbase

In the following paragraphs a brief review of the different linkbases will be given. And some code will be used to give an example of how linkbases in XBRL are constructed. The linkbases are built with the use of the XLink elements (see paragraph 9 of chapter I). The examples have originated from [What is XBRL?].

In fact, XBRL uses very little of the XLink element, from all the possibilities XBRL only uses 'locators' and 'from/to' in order to link all documents together.

3.1. Reference linkbase

In the Reference linkbase, the Financial Reporting Standard used is presented. A link is defined to the specific location where the specific term is defined within the Reporting Standard. In the example, the link is made to the IFRS of 2003. In “IAS” “1”, paragraph “66”, subparagraph “g” the information about ‘Cash and Cash Equivalents’.

```
<loc xlink:type=’locator’
  xlink:href=’ifrs-ci-2003-07-15.xsd#ifrs-ci_CashCashEquivalents’
  slink:label=’ifrs-ci_CashCashEquivalents’ xlink:title=’ifrs-
ci_CashCashEquivalents’/>

<reference xlink:type=’resource’
  xlink:label=’_ifrs-ci_CashCashEquivalents_link’
  slink:title=’ifrs-ci_CashCashEquivalents’>
  <ifrs-ci:Name> IAS </ifrs-ci:Name>
  <ifrs-ci:Number> 1 </ifrs-ci:Number>
  <ifrs-ci:Paragraph> 66 </ifrs-ci:Paragraph>
  <ifrs-ci:Subparagraph> g </ifrs-ci:Subparagraph>
</reference>
```

Example 1 – Reference

3.2. Labels linkbase

In the Labels linkbase it is possible to link a label from outside of the taxonomy with a terminology that is comprehensive for the reader in a future report. With the

use of the Labels linkbase it is also possible to give different language terminologies to the label. In the example code the English and Dutch synonym of the title “CashCashEquivalents” are placed.

```
<loc xlink:type='locator'
  xlink:href='ifrs-ci-2003-07-15.xsd#ifrs-ci_CashCashEquivalents'
  xlink:label='ifrs-ci_CashCashEquivalents'
  xlink:title='ifrs-ci_CashCashEquivalents'/>

<label xlink:type='resource'
  xlink:label="_ifrs-ci_CashCashEquivalents_link'
  xlink:title='ifrs-ci_CashCashEquivalents'
  xlink:role='http://www.xbrl.org/linkprops/label/standard'
  xml:lang='en'> Cash and Cash Equivalents </label>

<label xlink:type='resource'
  xlink:label="_ifrs-ci_CashCashEquivalents_link_nl'
  xlink:title='ifrs-ci_CashCashEquivalents'
  xlink:role='http://www.xbrl.org/linkprops/label/standard'
  xml:lang='nl'> Liquide middelen </label>
```

Example 2 - Labels

3.3. Presentation linkbase

For the reporting of financial facts the Presentation linkbase is used. In this linkbase the position of the financial asset in the report is defined. In the following example the post “CashCashEquivalents” is placed on the ninth position of the Current Assets post.
In the Presentation linkbase the Child-Parent combination is used.

```
<presentationArc xlink:type='arc'
  xlink:from='ifrs-ci_CashCashEquivalents'
  xlink:to='ifrs-ci_CurrentAssets'
  xlink:show='replace'
  xlink:actuate='onRequest'
  xlink:title='Go up to: CashCashEquivalents'
  xlink:arcrole='http://www.xbrl.org/linkprops/arc/child-parent' order='9'
use='optional'/>

<presentationArc xlink:type='arc'
  xlink:from='ifrs-ci_ CurrentAssets'
  xlink:to='ifrs-ci_CashCashEquivalents'
  xlink:show='replace' xlink:actuate='onRequest'
  xlink:title='Go down to: CashCashEquivalents'
  xlink:arcrole='http://www.xbrl.org/linkprops/arc/parent-child' order='9'
use='optional'/>
```

Example 3 – Presentation

3.4. Calculation linkbase

As the name already says, the Calculation linkbase is used to calculate the different subtotals of the financial report. Every post is given a weighting value of “1” or “-1” to define position of the value of the label on the balance. If this value must be added to the subtotal of the Parent, the value is “1”. If the value must be subtracted, the value must be “-1”. In the example, the “CashCashEquivalents” post has a weighting value of “1” and so it is added to the subtotal. Together with the other children of Current Assets this makes up the total value of the post.

```
<calculationArc xlink:type='arc'
  xlink:from='ifrsci_CashCashEquivalents'
  xlink:to='ifrs-ci_CurrentAssets'
  xlink:show='replace'
  xlink:actuate='onRequest'
  xlink:title='Go up to: CashCashEquivalents'
  xlink:arcrole='http://www.xbrl.org/linkprops/arc/child-parent' weight='1'
use='optional'/>
```

Example 4 - Calculation

3.5. Definition linkbase

Every link that is not represented in the Presentation linkbase is defined in the Definition linkbase. With this Definition linkbase it is also possible to make links with other taxonomies on an element level. With the use of this link it is possible to show that Reporting Standards between different taxonomies can be identical for this element.

An other functionality of the Definition linkbase, that does not influence the XBRL-processor at all but creates a more comprehensive picture for human beings, is the possibility to link two meanings that are similar with each other.

The XBRL International organisation has presented a document that explains how an organisation, branch or country can create their own taxonomy. To create these taxonomies, special taxonomy builders are available from several commercial suppliers.

3.6. Arc and Arcrole

As you can see in the examples of the Presentation linkbase and the Calculation linkbase the words “arc” and “arcrole” can be found. These so called Arcroles makes the connection between these two linkbases. For more information about arcroles see paragraph 9.2 of the XML chapter.

4. Instance

An XBRL instance basically is a linear collection of items, context and tuples. These subjects will be reviewed within this chapter. What you will see is that XBRL does not provide any means to restrict or guide the information captured in an instance. While XML provides in the tools to place restrictions on elements, XBRL chooses not to implement these possibilities. The result is that there is no format restriction on the elements, it could be possible to have two items with the same name with different values within an instance.

4.1. Namespace

An XBRL instance starts with the <xbrl> element and ends with </xbrl>. Within the first statement the basic parts of an xml file are defined such as the namespace and schema location. With XBRL, this is also the case. The opening tag usually contains quite a large list of namespaces. Let's start with an example from the XBRL 2.1 specification.

```
<xbrl xmlns="http://www.xbrl.org/2003/instance"
      xmlns:xlink="http://www.w3.org/1999/xlink"
      xmlns:link="http://www.xbrl.org/2003/linkbase"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:ci=”http://www.xbrl.org/us/gaap/ci/2003/usfr-ci-2003”

      xsi:schemaLocation="
      http://www.xbrl.org/us/fr/ci/2003/usfr-ci-2003
      http://www.xbrl.org/us/fr/ci/2000-07-31/usfr-ci-2003.xsd">

  <link:schemaRef xlink:type="simple"
     xlink:href="http://www.xbrl.org/us/fr/ci/2000-07-31/usfr-ci-2003.xsd"/>

  <ci:assets precision="3" unitRef="u1" ontextRef="c1">727</ci:assets>
  <ci:liabilities precision="2" unitRef="u1" contextRef="c1">635</ci:liabilities>

  <context id="c1"><!-- ... --></context>
  <unit id="u1"><!-- ... --></unit>

</xbrl>
```

Example 5 – XBRL instance example

These namespaces are all given an alias in order to increase readability for people and reduce the amount of text every time the namespace is used. It is not necessary since the processor of this file replaces every alias with the complete namespace. You can also write the entire namespace if desired. These namespaces are further discussed in chapter 5. After all the namespaces are given an alias, the schemaLocation is defined. While this is present in the example, its function has been replaced by the schemaRef element since XBRL version 2.1.

4.2. schemaRef element

The replacement of the schemaLocation element by the schemaRef element is part of the changes in XBRL 2.1. The complete set of taxonomy schemas and link bases supporting an XBRL instance has been defined as a Discoverable Taxonomy Set (DTS). This schemaRef element is obligatory; each XBRL document must at least have one of these elements. It should be placed as a child element of the XBRL element but before all other child elements. This tells the processor to what taxonomy this document adheres and where it is located. When using both a schemaLocation and a schemaRef, it should be checked if the information in both files is consistent. If they have inconsistencies, the processor may produce an error while processing. It is advised to make sure both files are consistent to prevent unexplainable errors or just leave the schemaLocation element out.
In this example, the schemaRef is the only element of the DTS, linking to the pgc-2005.xsd taxonomy. It is possible to link multiple taxonomies from a single document to use elements found within these taxonomies. Each of these links should be defined in a <link:schemaRef> element.

4.3. linkbaseRef element

What is not shown in the example from the XBRL specification is the linkbaseRef element. This element identifies a linkbase that becomes part of the DTS. It is not obligatory to use a linkbaseRef but when used, it should be placed directly after the schemaRef element and before any other elements in the document.

```
<link:linkbaseRef xlink:type="simple"
  xlink:href="http://www.someurl.com/calculation/calculation.xml"
  xlink:role=" http://www.xbrl.org/2003/role/calculationLinkbaseRef"
  xlink:arcrole="http://www.w3.org/1999/xlink/properties/linkbase" />
```

Example 6 – linkbaseRef notation

When using the linkbaseRef element, an xlink:type must be present and must have a fixed content “simple”. It also must contain an xlink:href attribute containing a URI. This must point to a linkbase that contains the extended links determined by the value of the xlink:role attribute. This xlink:role attribute can constrain the kinds of extended links that are permitted within the identified linkbase. There are several possibilities, shown in the table below.

Values of the linkbaseRef xlink:role **attribute**	Element pointed to by xlink:href
(unspecified)	MAY contain any extended link elements
http://www.xbrl.org/2003/role/calculationLinkbaseRef	MUST contain only calculationLink elements
http://www.xbrl.org/2003/role/definitionLinkbaseRef	MUST contain only definitionLink elements
http://www.xbrl.org/2003/role/labelLinkbaseRef	MUST contain only labelLink elements
http://www.xbrl.org/2003/role/presentationLinkbaseRef	MUST contain only presentationLink elements
http://www.xbrl.org/2003/role/referenceLinkbaseRef	MUST contain only referenceLink elements

Table 1 - Roles in the linkbaseRef element

The xlink:arcrole attribute must be included in this format to indicate that the linkbaseRef element points to a linkbase. When it is not included, a processor can not know the target of the linkbaseRef without accessing it. This would increase the processing time so the xlink:arcrole attribute is required, informing the processor that the target is a linkbase.

4.4. roleRef and arcroleRef element

Other optional elements in an XBRL instance are the roleRef and arcroleRef elements. The roleRef element is used to reference to definitions of any custom xlink:role attribute value used in footnote links in the XBRL instance. The arcroleRef element is used for the same purpose, only referencing to custom xlink:arcrole attributes in the footnote links instead of xlink:role attributes. When used, the roleRef element must be places directly after the linkbaseRef element. The arcroleRef element goes directly after the roleRef element.

4.5. Item element

Let's get back to example 5, the XBRL instance example. There are not only elements within an XBRL instance. There are also items, the real data within the document. These items represent a single fact or business measurement in the form of an abstract element. Because of this, an item will never appear alone in an XBRL instance. All items are part of a substitution group item of a substitution group based on item. These item elements must not be descendants of other item elements. Any structural relationships must be captured using tuples. Intellectual structure is captured by the linkbases instead of incorporating them within the XBRL instance.

```
<ci:assets precision="3" unitRef="u1" ontextRef="c1">727</ci:assets>
<ci:liabilities precision="2" unitRef="u1" contextRef="c1">635</ci:liabilities>

<context id="c1"><!-- ... --></context>
<unit id="u1"><!-- ... --></unit>
```

Example 7 – Items example

The value of the assets in this numeric context has the value “727” and the value of the liabilities has a value of “635”. Because these are numeric items, they must have either a precision or a decimals attribute unless it is of the fractionItemType, derived of this type or has a nil value. In those cases, it should not have either attributes. Any other numeric item must have one of the two attributes, not both. All other non-numeric items must not have a precision or decimals attribute.

4.5.1. precision attribute

In this example these precision attributes are given values of “3” and “2”, meaning that the value is know to be trustworthy for the purpose of computations. The value of the attribute identifies to what extend the value of the element is trustworthy. The value of a precision attribute must me a non-negative whole integer. In this example the first element has a precision attribute with a value of “3” indicating that, reading from left to right and ignoring any zero digits, the first 3 digits are trustworthy. An application that performs calculation with these figures should ignore any digits after the first 3 non-zero digits. By ignoring we don’t mean just using the first 3 digits but replacing all other digits with zeroes. So, if an application does a calculation with the assets, it should use the value “727”. If it does a calculation with the liabilities, it should use the value “630”.

4.5.2. decimals attribute

The decimals attribute must be an integer but can be negative. It specifies the number of decimal places that may be considered accurate as a result of rounding or truncating. If it has a value of “3”, the first 3 decimal places are known to be accurate. Meaning that an element with a value of “79” with a decimals attribute with a value of “3”, during calculations it should be regarded as “79,000”. If the element would be “79,0678” with a decimals attribute of “2”, it should be regarded as “79,06”. If the attribute is negative, “-3” for example, the value of the element is known to be accurate for all elements left of the third digit. An element with a numeric value of “12345” and a decimal attribute with value “-3” should be regarded as “12000”. If the decimal attribute would be “-1”, the value should be regarded as “12340”.

4.5.3. contextRef attribute

The elements have more attributes, the contextRef and unitRef attribute. All items must have a context; tuples must not have a context. This context is identified using the contextRef attribute. This indicates the context element associated with the item. The value of the contextRef attribute must be equal to an id attribute of the context element within the same XBRL instance. In the example, the items both have a contextRef with value “c1”. Further on in the document, a context

element is present with an id attribute with value “c1”. This context holds more relevant information about the facts in the item.

4.5.4. unitRef attribute

Besides a context, all numeric items must have a unitRef attribute. Non-numeric items and tuples must not have this unitRef attribute. As with the contextRef attribute, the unitRef attribute must have a value equal to the id attribute of a unit element within the same XBRL instance. In the example, the items have a unitRef with value “u1” and a unit element is present with id attribute “u1”. This unit element contains information on the units of measurement used.

4.6. Context element

As said before, the context element holds relevant information about facts presented in the items. It contains information about the entity itself, the reporting period and the reporting scenario. This information is necessary to understand the underlying business facts captured in the XBRL instance. The context element must adhere to a specific XML schema.

```
<schema targetNamespace="http://www.xbrl.org/2003/instance"
  xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:xbrli="http://www.xbrl.org/2003/instance"
  xmlns:link="http://www.xbrl.org/2003/linkbase"
  elementFormDefault="qualified">

  <element name="context">
    <annotation>
      <documentation>
      Used for an island of context to which facts can be related.
      </documentation>
    </annotation>
    <complexType>
      <sequence>
        <element name="entity" type="xbrli:contextEntityType" />
        <element name="period" type="xbrli:contextPeriodType" />
        <element name="scenario" type="xbrli:contextScenarioType" minOccurs="0" />
      </sequence>
      <attribute name="id" type="ID" use="required" />
    </complexType>
  </element>

</schema>
```

Example 8 – Context element XML schema

4.6.1. id attribute

Every context element must contain an id attribute; this gives the context a unique name that can be referenced to by item elements with a contextRef attribute. This id attribute must conform to the XML rules for attributes with the ID type and it is required not to start with a number.

4.6.2. period element

The period element is used to define the time referenced by an item element. This can be a time span with a starting and ending date(startDate and endDate), a point in time (instant) or it can be forever (forever). These options have requirements, all specified within another XML schema. If an item element has a periodType=“instant”, the period element must contain an instant element. If the item element has a periodType=“duration”, it must contain either forever of a valid sequence of startDate and endDate. This sequence can be defined but when it is not specifically defined, the date elements take on default values of startDate and endDate. The startDate will be the current date and a time part of T00:00:00, the nextDate will be midnight of the same day, the current date plus P1D with timepart T00:00:00. This is because a value of 24 is not allowed. When the endDate is supplied, it must specify a point in time later then the specified or implied startDate.

4.6.3. entity element

Another obligatory part of the context element. This describes the entity of the fact captured in the item, whether it is a business, individual, government department, etc. This element must contain an identifier element. This identifier identifies a scheme for identifying business entities. This scheme attribute contains the namespace URI of the identification scheme. The value of the identifier element must be a token that is valid within the specified scheme. This is not checked by XBRL since it is not a naming authority for business entities so be sure it is valid.

The second element within the entity element is the optional segment element. This is used as an add-on where the entity identifier is insufficient. The elements used must not be part of the XBRL defined namespace at “http://www.xbrl.org/2003/instance”. They also must not be in the substitution group of elements defined in this namespace. However, it also must not be empty when used. If you use this element you must provide the proper namespace support to ensure that a XML schema validator can properly validate the segment element.

The final, optional, element is the scenario element. As with the entity element, it must not be defined within the XBRL instance namespace and when used must not be empty. This element is used to document the circumstances during the measurement of the business facts captured in the XBRL instance. This can be things as actual, budgeted, estimated and more. As long as a proper namespace is provided, anything is possible for internal use.

4.7. unit element

The unit element gives information about the measured units of a numeric item. This content must be a simple unit with a single measure element or a ratio of products of units of measure. This ratio is represented with a divide element containing a numerator and denominator.

4.7.1. id attribute

Each unit element must have an id attribute, just as the context element this gives a point of reference for item elements in the instance. This id attribute must conform to the XML rules for attributes with the ID type and it is required not to start with a number.

4.7.2. measure element

This element is of the type xsd:QName. Some facts have restrictions on the content of the unit element and the value of the measure element. This is because of the type of concept they represent. The monetaryItemType of derivatives of it must have xsd:QName="http://www.xbrl.org/2003/iso4217". This contains an ISO 4217 currency designation that was valid during the time period in which the measurements have taken place. Items of the type sharesItemType and derivatives must have xsd:QName="http://www.xbrl.org/2003/instance".
Rates, percentages and ratios must be reported with decimal or scientific notation, not with values multiplied by 100. The original values must be used and the prefix used must resolve to the "http://www.xbrl.org/2003/instance" namespace. Complex items can be used but must be expressed with simple items, multiplication of measured elements combined with a divide element.

4.8. Tuples

As some business facts can only be understood if they are combined with another fact, a tuple exists. Most of the time, each and every business fact makes for one independently understandable fact. A tuple is a set of facts, combined. Tuples are of great value when a fact is repeated multiple times, so the individual occurrences can easily be seen separately.

Tuples also have strict rules to adhere to, as they consist of complex content. A tuple may contain both items and other tuples, and is like the item-element an abstract element. The following rules apply to tuples and consequently to their declarations in taxonomy schemas:
Tuples and the children of the tuple must be members of the substitution group that has tuple as its head, which makes the tuple to be declared globally as only global items can occur in a substitution group.
A tuple cannot contain periodType and/or balance attributes, nor can they or the tuple definitions in taxonomy schemas contain or permit either mixed or simple content.
As tuples would be referenced from elsewhere, the specification of an id attribute is recommended, but not required. If this attribute is not specified, the tuple cannot be referenced by shorthand xpointers.

Tuple declarations in taxonomy schemas should not, but are prohibited to, specify local attributes, other than the id attribute mentioned in the previous rule.

An example of a tuple is the following, copied from the reference-manual.

```
<s:managementInformation>
  <s:managementName contextRef="c1">Haywood Chenokitov</s:managementName>
  <s:managementTitle contextRef="c1">President</s:managementTitle>
  <s:managementAge precision="2" contextRef="n1" unitRef="u1">42</s:managementAge>
</s:managementInformation>
<s:managementInformation>
  <s:managementName contextRef="c1">Miriam Minderbender</s:managementName>
  <s:managementTitle contextRef="c1">CEO</s:managementTitle>
</s:managementInformation>
```

Example 9 – Tuples

As tuples can themselves contain other tuples and other complexContent, a layered structure can be created.

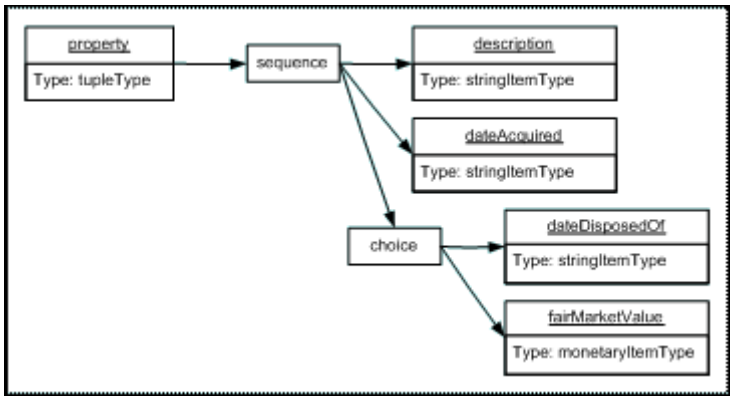


Figure 2 – Hierarchy in a tuple.

The content models for tuples can be defined using only XML Schema; content models for tuples are not defined or modified by any of the XBRL linkbases.

4.9. Footnotes

Besides the relations depicted in tuples, non-fixed relations between elements also can be found in quite a few XBRL-documents. These irregularly structured associations can also be embedded in an XBRL-document by the use of the footnoteLink element.
This element can contain up to five elements: title, documentation, loc, footnoteArc and footnote, of which we will discuss some in the following paragraphs.
The next example is also taken from the definition-manual.

```
<link:footnoteLink
  xlink:type="extended"
  xlink:title="1"
  xlink:role="http://www.xbrl.org/2003/role/link">
  <link:footnote
    xlink:type="resource"
    xlink:label="footnote1"
    xlink:role="http://www.xbrl.org/2003/role/footnote"
    xml:lang="en">Including the effects of the merger.</link:footnote>
  <link:footnote
    xlink:type="resource"
    xlink:label="footnote1"
    xlink:role="http://www.xbrl.org/2003/role/footnote"
    xml:lang="fr">Y compris les effets de la fusion.</link:footnote>
  <link:loc xlink:type="locator" xlink:label="fact1" xlink:href="#f1"/>
  <link:loc xlink:type="locator" xlink:label="fact1" xlink:href="#f2"/>
  <link:loc xlink:type="locator" xlink:label="fact1" xlink:href="#f3"/>
  <link:footnoteArc
   xlink:type="arc"
   xlink:from="fact1" xlink:to="footnote1"
   xlink:title="view explanatory footnote"
   xlink:arcrole="http://www.xbrl.org/2003/arcrole/fact-footnote"/>
</link:footnoteLink>
```

Example 10 – Footnotes in different languages

The footnoteArc element connects, very similar to a regular arc-element in XML, several elements to each other. This footnoteArc connects 2 footnotes to three elements.

4.9.1. Locators

The different locators within the footnoteLink element have to adhere to some rules; they have to be loc elements. Also, the loc element can only point to items or tuples in the same XBRL instance that contains the loc element itself.

4.9.2. Footnote element

The footnote element is the only allowed resource in a footnoteLink element. A footnote element may have mixed content containing a simple string or XHTML, or a mixture of both.

A footnote element has only one predefined role, which is: “http://www.xbrl.org/2003/role/footnote”. Furthermore, every footnote element is obliged to have an xml:lang attribute specifying the language used for the content of the specific footnote.

4.9.3. footnoteArc element

A footnoteArc element has the same syntax as a generic extended link arc, which can be found in the XBRL specification. Being a link arc, it has to have a role-attribute and all cases for the footnoteArc element this will be:
“http://www.xbrl.org/2003/arcrole/fact-footnote”.

This element also has the optional title attribute which has to have, when used, a string as its value. The title can be used to give information about the relationship between facts and related footnotes to users navigating between those facts and footnotes.

5. Namespaces revisited

As mentioned in the previous paragraph an XBRL-instance contains a lot of references to various namespaces to define the “worlds” of elements which can be used in the instance. Each namespace is basically a logical group of element specifications, made for a specific, distinct part of XBRL. Such a namespace is then contained in a schema (taxonomy) which contains the actual definitions. Though it is possible to include all namespaces and element definitions into one taxonomy, it makes sense to distinguish various parts:

- each special-purpose namespace/taxonomy can be managed by a different task force of experts: they do not get in each other’s way when releasing a new version or naming elements and attributes
- for an XBRL-instance creator it is possible to combine different namespaces/taxonomies into one complete set, according to his special needs
- various XBRL-standards exist for various purposes, but some parts are equal for all: these parts can be contained in a namespace/taxonomy which does not change as frequently
- for people writing or reading an XBRL-instance it is convenient to have a logical grouping of definitions which belong to each other and can be identified with their own namespace

In this paragraph the namespaces typically used in an XBRL-instance will be described: XBRL-Instance (xbrli), ISO-4217 (iso4217), Linkbase (link), XLink (xl), XMLSchema-instance (xsi) and USA-GAAP (ci).

5.1. XBRL-Instance

The XBRL-Instance (XBRLI) namespace is meant for containing the basic structure (elements, attributes) and data types of an XBRL-instance document. The namespace of the XBRL-Instance is called: “http://www.xbrl.org/2003/instance”. Since a namespace is just an identifier, it does not have to point to a real URI. Instead the schema for this namespace for the XBRL 2.1 specification can be found at: “http://www.xbrl.org/2003/xbrl-instance-2003-12-31.xsd”.

So, in short: the definition of the XBRL-Instance namespace for XBRL 2.1 is contained in the schema: “http://www.xbrl.org/2003/xbrl-instance-2003-12-31.xsd”. Paragraph 4 gives a detailed description of how an XBRL-instance is structured and which elements can be used for what purpose. For this reason this paragraph will only give a brief overview of what the purpose is of the xbrli namespace and how it is defined in the accompanying schema.

First of all the XBRLI-schema imports another schema: “xbrl-linkbase-2003-12-31.xsd”. This schema is imported, concerning the namespace: “http://www.xbrl.org/2003/linkbase”. So the linkbase-schema is imported, but not

used in the XBRLI-schema itself. It is now possible to use the Linkbase-schema in an XBRL-instance without referencing it separately.

Second, the XBRLI-schema defines the structure of an XBRL-document. This structure is presented in figure 3 on the following page:

- “xbrl” is the root element of the instance
- The xbrl-element can contain a custom number of attributes, of which the id-attribute is an optional one.
- Below the xbrl-element at least one schemaRef-attribute should be included.
- Then a number of linkbaseRef, roleRef and arcroleRef attributes can be included.
- Items and tuples can be included, which will contain the actual data of the financial report. Item and tuple are abstract elements, so they will not actually be used in an instance, but be replaced with other elements.
- Context, unit and footnotes can be included.

As mentioned earlier, an elaborate description of these elements can be found in the previous parts.

Third, the XBRL-schema defines data types which can be used in the factual elements of an instance. For example: periodType, balance, monetary and shares.

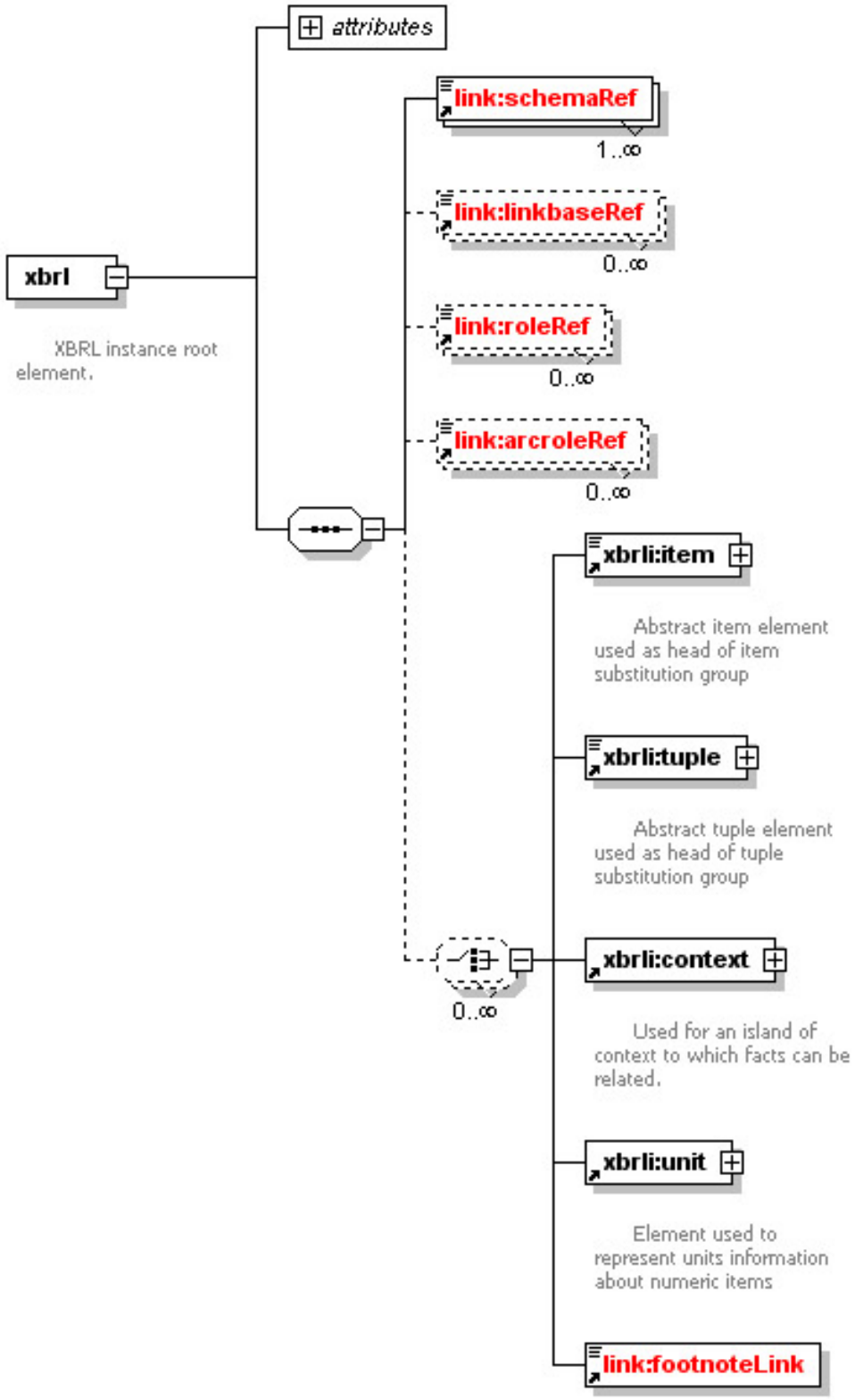


Figure 3 – Basic structure of an XBRL-instance

5.2. ISO-4217

This namespace is used for containing the currency codes as specified by ISO (International Standards Organisation). The namespace is called: "http://www.xbrl.org/2003/iso4217". These currency codes are needed since for facts concerning money need it is necessary to specify in which currency the money is presented.
The schema containing the definitions for this namespace could not be found.

5.3. Linkbase

The Linkbase-namespace, usually given the alias "link", is meant for describing how linkbase elements can be made in an XBRL-instance document. The name of this namespace is: "http://www.xbrl.org/2003/linkbase". See paragraph 3 for an elaborate description of linkbases. The schema which belongs to the linkbase-namespace for XBRL 2.1 is: "http://www.xbrl.org/2003/xbrl-linkbase-2003-12-31.xsd".

The general structure of a linkbase element is presented in the following figure:

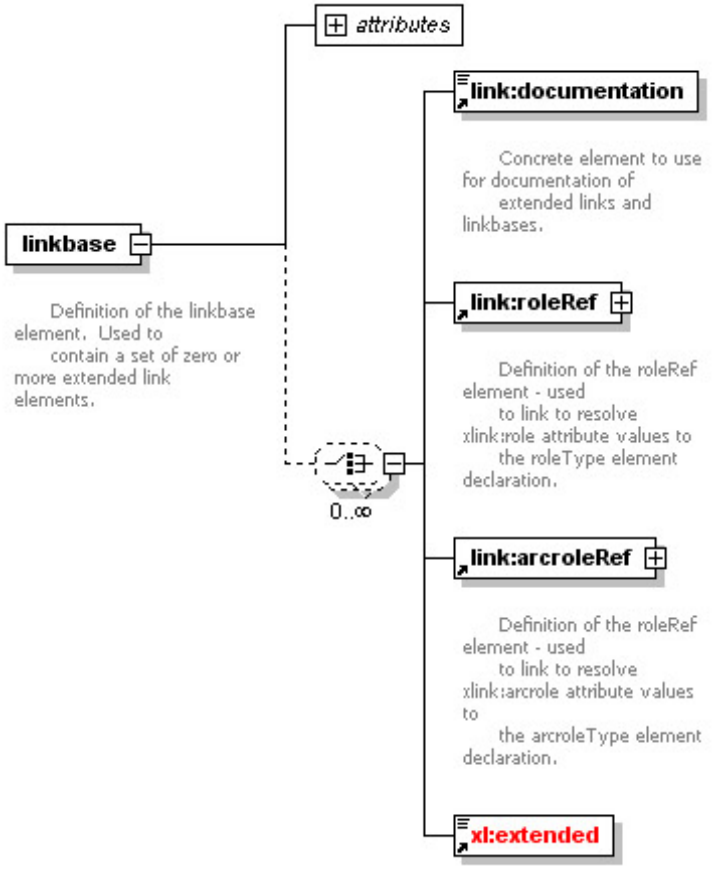


Figure 4 – Structure of the Linkbase-element

5.4. XLink

XLink is a W3C recommendation as of June 27 2001. The recommendation is a specification for how to construct links between XML-instances. Since the specification lacked some features needed for XBRL, an XBRL-specific extension was made. The name of the namespace is: “http://www.w3.org/1999/xlink”. The XBRL 2.1 schema concerning this namespace can be found at: “http://www.xbrl.org/2003/xlink-2003-12-31.xsd”.

5.5. XMLSchema-instance

An XBRL-instance document can make a reference to a schema in two ways: by using the schemaRef-element, which is specifically made for XBRL (see chapter 4.2) or by using the W3C standard for XML Schema’s. It can also do both, but then the referenced schema’s should match to prevent errors from an XML parser when the instance document is parsed. When referencing a schema in the conventional way, using the W3C recommendation, it is necessary to use the XMLSchema-instance namespace. The schema relating to this namespace defined how the schema should be referenced, so a parser can understand the reference.

The name of the XMLSchema-instance namespace is: “http://www.w3.org/2001/XMLSchema-instance”. The schema for this namespace can be found at: “http://www.w3.org/TR/xmlschema-1”.

5.6. US-GAAP

The namespaces in the previous paragraphs are generally applicable for XBRL and used in virtually all XBRL DTS’s (Discoverable Taxonomy Sets). The US-GAAP namespace is a special namespace for defining how an XBRL-instance document should be structured to adhere to the US GAAP accounting rules.

Since different sets of accounting rules exist, also different DTS’s exist which relate to this accounting rules. XBRL International recognizes two types of taxonomies (taxonomy sets): approved and acknowledged. The approved taxonomies adhere to the XBRL specification and the XBRL Guidelines. The acknowledged taxonomies only adhere to the XBRL specification. Besides these taxonomies also a GL-taxonomy exists, which is a general taxonomy. This taxonomy is meant as a base for internal reporting or as a base for constructing specific taxonomies such as US GAAP.

To give an idea of existing taxonomies, the following taxonomies are approved by XBRL International:

- US GAAP - Commercial and Industrial
- US GAAP - Banking and Savings Institutions
- US GAAP – Insurance
- US GAAP - Investment Management
- SEC Certification
- Management Report
- Accountants Report
- MD&A

Paragraph 2 contains more information about taxonomies.

5.7. Overview

To give a quick graphical overview of the myriad of documents, taxonomies, instances, namespaces and schemas, we've developed a cheatsheet listing the important entities within XBRL.

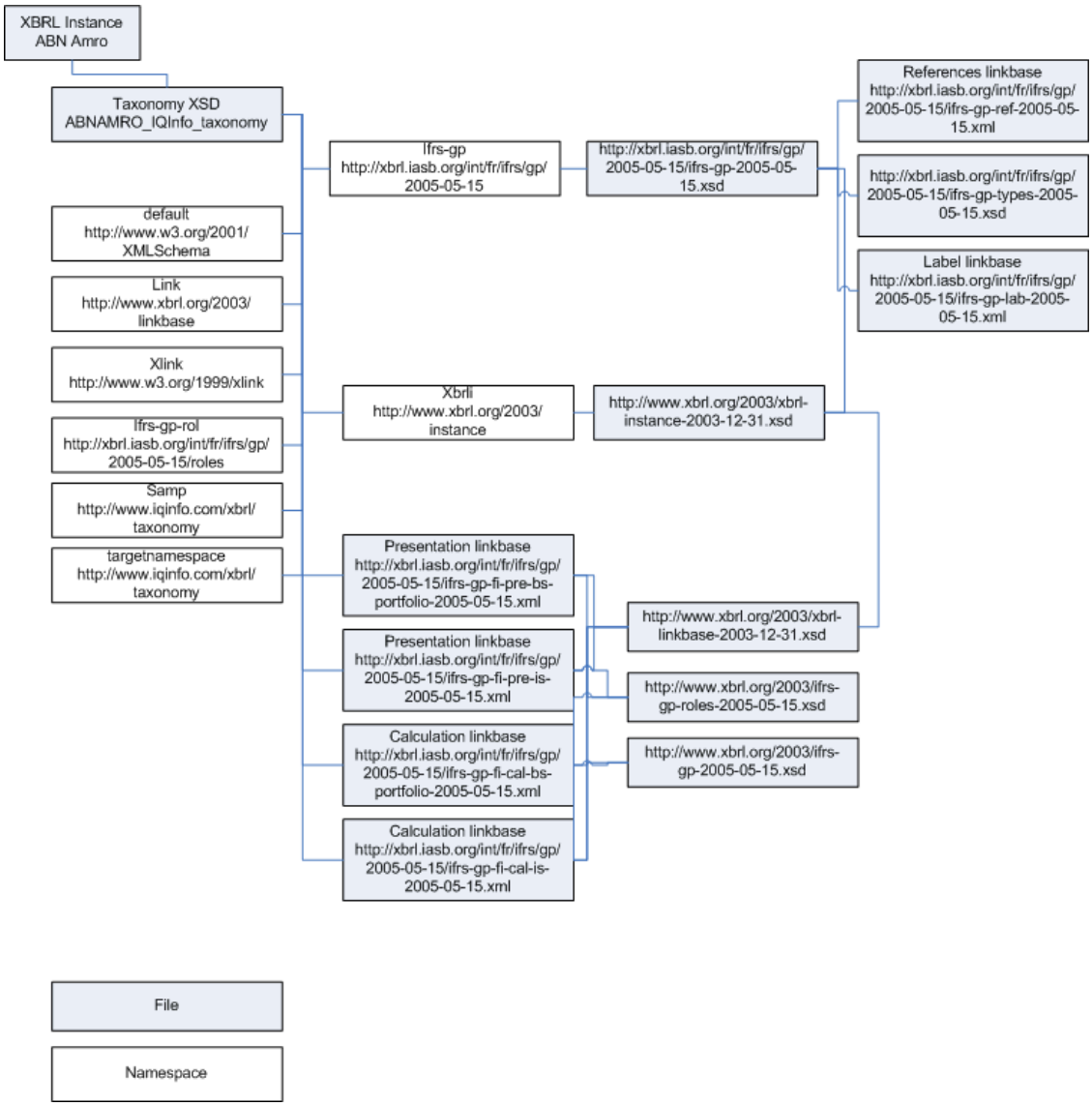


Figure 5 – The cheatsheet

6. XBRL supporting documents

6.1. FRTA

The Financial Reporting Taxonomy Architecture (FRTA) is a document published by the XBRL International Consortium. It recommends architectural rules and proposes conventions that assist authors in creating taxonomies. It is aimed at creating documents that give better performance with processors that can be used among different financial reporting taxonomies.

6.2. FRIS

The Financial Reporting Instance Standards (FRIS) is also published by the XBRL International Consortium. This document is intended to complement the FRTA document. It aims to facilitate the analysis and comparison of XBRL financial reporting data by either processors or humans. These financial documents are intended to satisfy financial reporting standards and accounting principles. The FRIS is not intended for use on other types of documents, like journal-level reporting or narrative reports and other non-financial data.

IV. A short overview of NTP

1. Introduction

This chapter tries to shed some light into the dark world that to some is known as the NTP-project. NTP stands for 'Nederlands Taxonomie Project' (Dutch Taxonomy Project) and it's homepage can be found at http://www.xbrl-ntp.nl.
This foundation has been set up to create a taxonomy to be used by the organizations which have to pay taxes in the Netherlands, this taxonomy enables them to send their information to the tax authority ('Belastingdienst') and the chambers of commerce ('Kamers van Koophandel' abbreviated to 'KvK').

We will explain the basic setup of the NTP, give an introduction to the NTP and it's organization and will eventually get to grips with a report to show it in more detail.

2. NTP

XBRL, as we all know, tends to get managers pretty excited about it's potential and the possibilities of it. Coherent and standardized reporting throughout all businesses and markets, both locally and globally, it sounds too good to be true.
Unluckily, this seems to be the case, although not completely.

Recent studies on the amount of money that can be saved when XBRL is utilized at it's fullest potential show savings up to 350 million Euro for the Dutch government, while the administrative burden on companies in the Netherlands should be lowered by about 25 percent which roughly equals 4 billion Euro each year as of 2007.
So far, so good. The Dutch government has legally ordained all companies in the Netherlands to electronically send their information in an XBRL-document to both the tax authority ('Belastingdienst') and the chambers of commerce ('Kamers van Koophandel' abbreviated to 'KvK'). As of now only very few have already handed in their information in this format due to several reasons, of which the most interesting is stated below.
There have been complaints about XBRL being too hard to understand and, ironically, the high costs which come with the successful transformation of the company-information into XBRL.

As XBRL is eXtensible, quite a few extensions have been created. A very big and dominant addition has been the creation of the IFRS-standard. 'International Financial Reporting Standards' are the standards companies in the US, Europe and Russia have to adhere to when creating reports for instances like the tax authority or the stock-exchange authority.
This IFRS-standard is very large as it has to accommodate all possible firms in all possible countries, languages and jurisdictions. Quite a few countries have therefore devised their own taxonomies, in the Netherlands this is NTP.
NTP stands for 'Nederlands Taxonomie Project' (Dutch Taxonomy Project) and uses the IFRS-standard as building blocks to create reports and taxonomies specifically designed for firms housed in the Netherlands, completely adhering to the Dutch law and several other specific requirements.

The following chapter will elaborate more on the building blocks and the structure of the NTP, and how IFRS is being used to create 'scoped' reports.

3. Basic structure

The NTP is created for three different situations. These three branches of the NTP taxonomy are focussed on the Chamber of Commerce, the Dutch Tax authority and the Dutch Statistical Bureau. Each of these focuses have their own implementation in NTP. Further is the NTP build upon the existing XBRL representation of the Accounting standards of IFRS that are imported into NTP. NTP is thus an extension of XBRL.

The Dutch Taxonomy created in the NTP consists of three stages. The first being the 'Basis' consisting of accounting standards like i.e. IFRS in XBRL that are through this basis layer being imported. The second being the 'Domein' in which, per domain from the three of the aforementioned branches of the NTP Taxonomy, the relations between the elements on the annual account and the to be published report are presented. The third stage is the 'Report' level in which the different to be published posts are defined for the Chamber of Commerce, the Dutch Tax authority and the Dutch Statistical Bureau.
The basic structure of the NTP taxonomy can be better understood with the use of the following image from 'Reducing administrative burdens through standardisation'. In this image the complete architecture of this Taxonomy is represented from the basis, importing e.g. the IFRS on the bottom layer to the report layer on top.

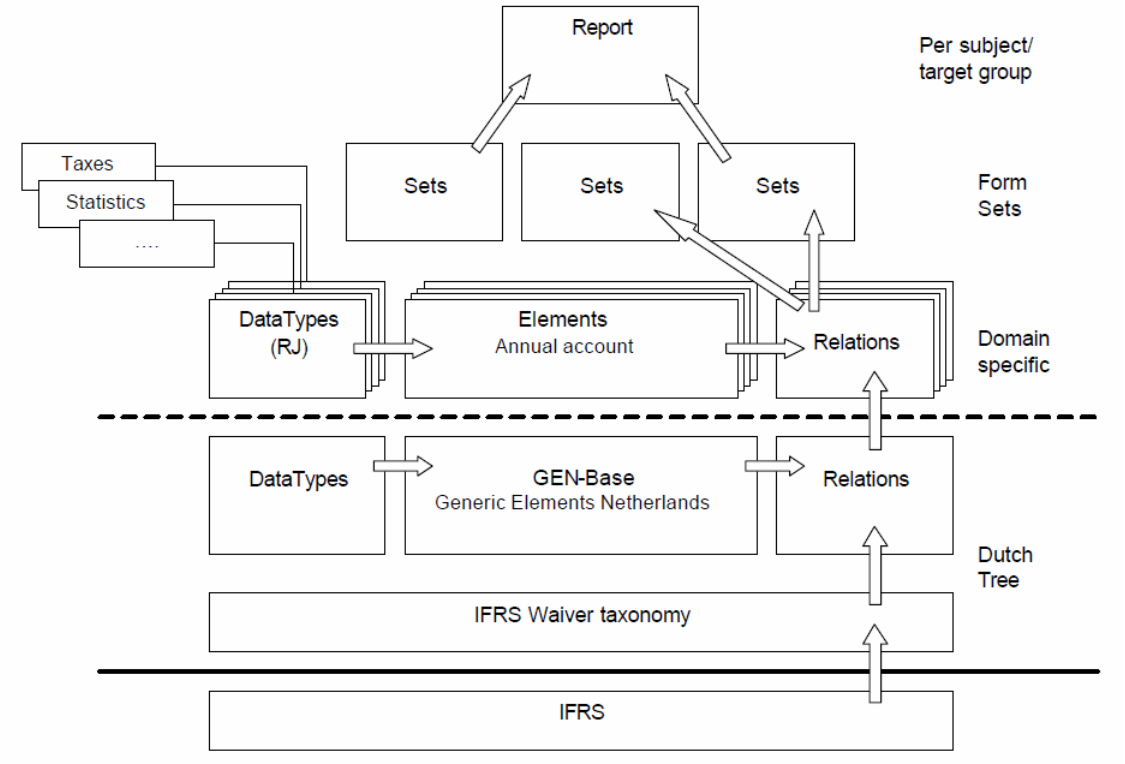


Figure 1 – A schematic view of the NTP

In the second figure (below) a more abstract view of the architecture of the NTP is given. This image shows the basis layer with e.g. IFRS, with on that three pillars of the 'CBS', 'KvK' and 'bd', each having their own 'domein' and formsets to come to the reports.
Because these images do not fully reflect the representation of the NTP, we will in the next pages examine a NTP report in more detail.

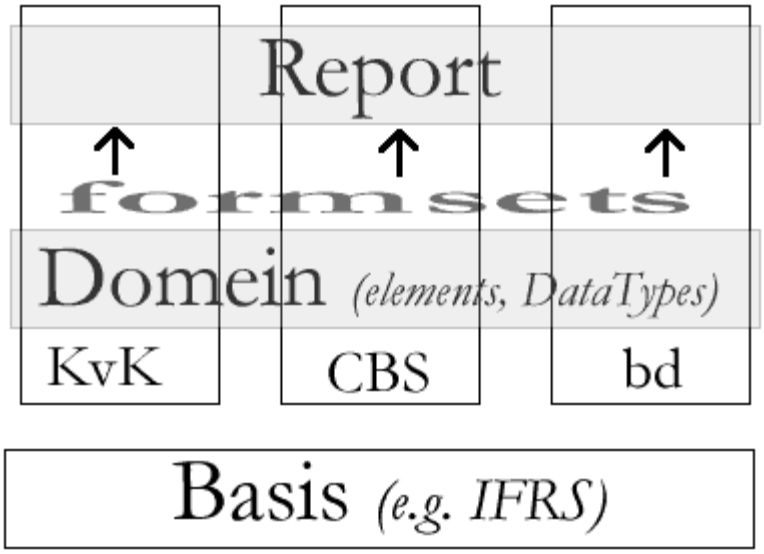


Figure 2 – A more abstract view of the NTP

4. Examining an NTP-report

In this paragraph we will take a closer look at one of the reports that is being used within the NTP. It has no specific name, but is referenced to as ‘rpt-kvk-balansd-2006.xsd’.
As can be concluded from the name of the file, this is a report belonging to the KvK-formset and displays the balance. This balance is defined in 2006.
This report is actually the most commonly used report as it is especially intended to be used by small companies (1 to 10 employees) to report their balance to the ‘kvk’.

The NTP consists of several ‘packages’, each consisting of vital parts to the complete NTP-taxonomy. When taking a closer look at the complete NTP-package, it is clearly divided into 3 main areas: ‘basis’, ‘domein’ and ‘report’. These can be translated into Basis, Domain and Report.
Each area has several subcategories, for which the subcategories for Domain and Report are the same. These are: ‘kvk’, ‘bd’ and ‘cbs’, and are abbreviations for the chambers of commerce, tax authority and statistics bureau respectively.
Basis consists of more subcategories, all either linking to other resources such as the IFRS-taxonomies or specifying reusable content such as common elements (Strings, monetaryTypes and so on. Please refer to the known literature for further explanation of these elements.)

The ‘kvk’ sub-directory of Domain is also set up with several subcategories, ‘common-data’, ‘rj’ and ‘formsets’. The latter is the most interesting as it was created to tackle a major problem in XBRL and XML: how to exclude items from being automatically inherited in an import.
Every time a schema is imported into another all elements become part of the Discoverable Taxonomy Set (DTS) and can be used in any schema or taxonomy importing this first schema. Most of the time this behaviour is intended and comes in really handy, but sometimes the author of a taxonomy would like to restrict access to some elements of another taxonomy as of possible reuse of the intended subjects or various other reasons.
XML and thus XBRL has no neat way of excluding elements or complete namespaces; when a single element or namespace is imported it can be used throughout the ‘child-taxonomies’.

The NTP has come up with quite an interesting solution to this by creating so called formsets. A formset is a small and specific import of several elements and/or namespaces in order to leave all unused elements out of the scope.
A small example is posted below.

This snippet of code is taken from the formset directly belonging to the report we’ve chosen to elaborate on. The code clearly shows one element (“ifrs-gp_AssetsNonCurrentTotal”) being imported and related to in a presentationArc.
The file we’ve quoted from is filled with imports such as these, 35 in total.

```
  <loc xlink:href="http://xbrl.iasb.org/int/fr/ifrs/gp/2005-05-15/ifrs-gp-2005-05-
15.xsd#ifrs-gp_AssetsNonCurrentTotal" xlink:label="ifrs-gp_AssetsNonCurrentTotal"
xlink:type="locator"/>
    <presentationArc order="4" preferredLabel="http://www.xbrl.org/2003/role/label"
use="optional" xlink:arcrole="http://www.xbrl.org/2003/arcrole/parent-child"
xlink:from="ifrs-gp_AssetsNonCurrentPresentation" xlink:to="ifrs-
gp_AssetsNonCurrentTotal" xlink:type="arc"/>
```

Example 1 – Snippet from fs-kvk-balansd-2006-presentation.xml

By just importing the 35 elements, the remainder of elements can be ignored so to speak by the XBRL-interpreter which makes up for two things: faster processing and easier creation and validation of an instance.
Unfortunately, the NTP taxonomies don't utilize this approach to the fullest as not only references to the formsets are used, the entire IFRS-taxonomy is also imported in a previous stage.
This way the DTS not only consists of the needed elements, but it also has all elements in it, 5488 in total; rendering the quite clever way of using the formsets as filters useless.

Without going too deep into details of this one report, we would like to point out the very modular way of constructing a taxonomy. Each and every taxonomy builds upon a taxonomy previously defined, be it by the IFRS, the NTP or US-GAAP: all are being used as building blocks for other taxonomies.
This is the strength of XBRL, and probably the reason why it will fail to be very successful; it's versatility will also be it's doom. As the taxonomies get to a lower level of inheritance, the intricacy of the taxonomy increases too. At some point it will overwhelm the proposed user with information, rendering him or her helpless and ultimately leading to the user not using XBRL.
In the Netherlands, companies are obliged to use XBRL for all external reporting to official organizations but we feel the user-unfriendliness will not invoke an immediate desire to also do internal and other external reporting in XBRL with the company.

We've added two images of the report below, one in Dutch and one in English. We've added both of them to show how easily the language can be switched as information about both languages is provided in the 'label'-linkbase.
This linkbase holds all element-names and provides the presentation-linkbase with all needed languages and alternative names.

The images below have been created using the Taxonomy Viewer, as created by Semansys.

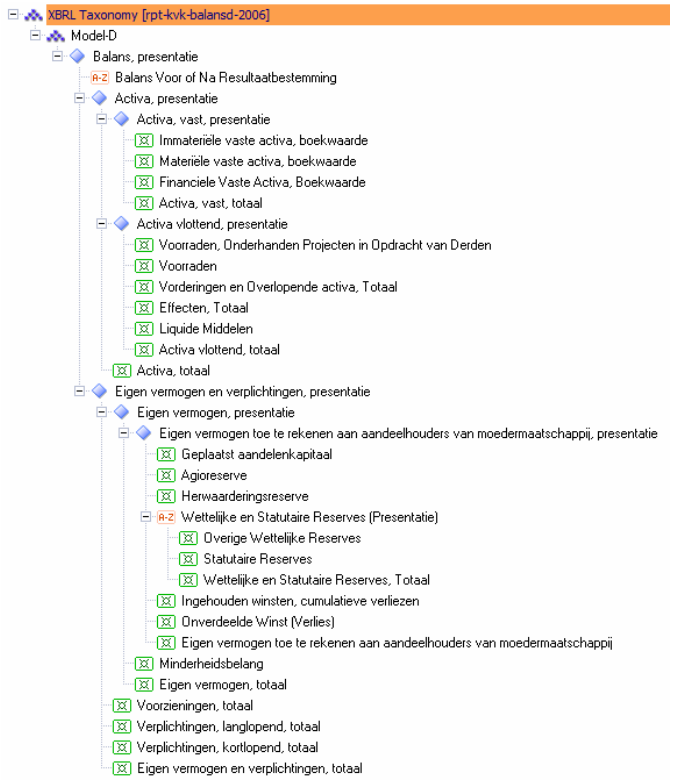


Figure 3 – The 'rpt-kvk-balansd-2006.xsd' in Dutch

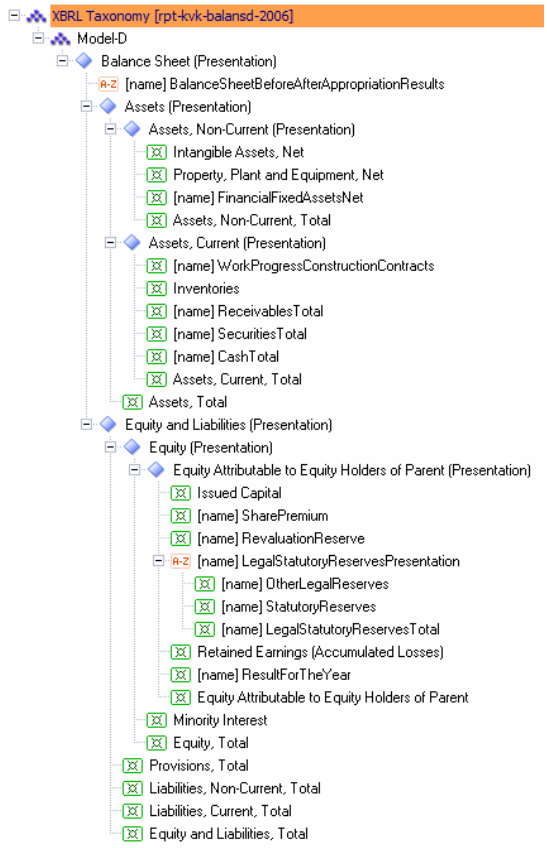


Figure 4 – The 'rpt-kvk-balansd-2006.xsd' in English

Returning to the topic of formsets, a formset is actually a mini-taxonomy which imports several (small) namespaces, but has also a presentation-linkbase linked to it. This linkbase is the linkbase we've quoted from in example 1, and is the one responsible for the actual selection of the needed elements.

This added level of imports does not make XBRL easier, it just adds to the intricacy of XBRL in general and specifically the NTP. We couldn't help noticing the amount of work to be done to gain a sufficient level of knowledge about XBRL to use it in a meaningful way.
As all firms in the Netherlands need to do all external reporting with XBRL-documents within the next few years, a lot of knowledge will have to be exchanged for them to grasp what they are doing. Due to the way XBRL was designed, this amount of knowledge to report in XBRL is quite low, as most of the work in reporting is just filling in the fields in an XBRL-instance, not creating entirely new taxonomies or editing them to fit your organization perfectly.
However, we foresee a large market in XBRL-consulting and a lot of new, small companies assisting large companies in making the change to both internal and external reporting in XBRL.

V. Ontologies, RDF and OWL

1. Introduction

The goal of this paragraph is to give the reader a basic understanding of the Web Ontology Language (OWL) and the concepts, languages and standards surrounding this language. After reading this section and the previous paragraphs about XML and XBRL, the reader understands enough of XML, XBRL and OWL to begin exploring how these techniques can be brought together for creating ontologies about business reporting.

This section is structured as follows: paragraph 2 contains an introduction to the Semantic Web, a new form of our currently known World Wide Web. Then paragraph 3 contains information on ontologies, paragraph 4 is about RDF and RDF-Schema and paragraph 5 delves into the details of OWL.

2. Semantic Web

The Semantic Web is a concept which is used to identify a new form of the currently existing World Wide Web. The current World Wide Web is mainly only human-understandable: humans can read a Web document and understand its contents. For example: a product catalogue page of Amazon.com; a human understands that the page contains names of books, with their description, price and shipping time. He also knows that a book consists of pages, is written by one or more authors, that the price can be stated in euros or dollars, etc. Computers cannot understand this: they can only “see” that one page links to another page (or media file) and present us with a link. Also the way a computer lets us find information on the World Wide Web is fairly basic: it crawls all pages, creates an index of the content and lets us search through them using some keywords. Although the search algorithms used today are fairly advanced, they are no match for the understanding capabilities when a human reads through the documents (though much slower of course).

This is where the Semantic Web can help: it provides a means to let computers “understand” the meaning (= semantics) of information. This “understanding” should not be thought of as very advanced though: the meaning of one piece of information (subject) is described with typed properties (predicate) to other pieces of information (object). Together all these subjects, predicates and objects create a web through which a computer can find his way: a Semantic Web. Think of a Semantic Web as the World Wide Web with all its documents and hyperlinks linking to each other, but then the hyperlinks are typed: instead of just linking the meaning of the link is also given and the links are not between documents, but between much smaller bits of information, namely subjects (the concept something is stated for) and objects (the related concept).

Since a computer cannot really understand and learn the way humans can, it has to be told first how certain types of subjects and objects (classes) can relate to each other. For example, it has to be told that a city contains many streets and zip codes and that these together, with a street number added, can form an address. And that a difference can be made between postal and home addresses, and that it makes no sense to send a pizza delivery to a postal address, and so on and so on. Together these descriptions of how things are related to each other and inherently what they mean are called Ontologies. These ontologies are discussed in chapter 3.

So in a Semantic Web computers can now search through great amounts of data and understand the meaning of the data, by using an ontology. Ontologies can also be linked together to create even larger ontologies. Since an ontology will always deal with a certain limited domain of concepts, no ontology will be complete or true for everyone. But when used (alone or combined) they can provide us with more relevant information than a traditional search engine on the traditional World Wide Web ever could.

Since a computer can understand the data much better now than was possible before, it can do much more intelligent search work for us. For this the concept of an “agent” is introduced: this is a piece of software that takes a complex question or request from a human and goes on its way through the Semantic Web to find the best result. For example a question like: “I would like to *buy* an *introductionary book* about *politics*, should *cost* me no more than *40 euros*, have *good reviews* of *recognized book reviewers* and be *delivered within two days*.” The words in *italic* are concepts the agent should understand: know the relationship to other concepts and what this relationship is. Using (combined) ontologies and some form of intelligence (reasoning) of its own the agent could figure out that books can be bought, has one or more subjects, has a price for which a limitation is given and has reviews by reviewers who can have a better or worse reputation (synonym for recognized). The agent also has to figure out that another limitation is given about the book to be bought, namely that it should be delivered within two days. When the agent “understands” the request it is given, it can now go searching through the Semantic Web to find the best available match and present it to the user (the human).

3. Ontologies

As mentioned in the previous part, ontologies represent a dataset of concepts within a certain domain and all the relationships between those concepts. Because of the linkage between the concepts it becomes possible to reason with this information. This chapter describes the elements of an ontology and the basic layout.

An ontology consists of a few basic elements, these are instances, concepts (also known as classes), attributes and relations. An ontology with a set of individual instances and classes is known as a knowledge base. You may argue that the ontology could also contain these instances and classes. There are no real rules that tell when something should be called an ontology and when it becomes a knowledge base. We will focus on the ontology and its elements.

3.1. Instances

The instances form the basis of an ontology, they are the lowest components available within an ontology. These instances are real-life objects, such as a car or a person or animal, but also abstract objects like a word or a number.

3.2. Concepts

The next component in an ontology is a concept. This is an abstract collection of instances, concepts or both. Because a concept can contain another concept things can get very vague and complicated, it is even possible for a concept to be part of its own concept. To prevent these kinds of complications it is possible to have restrictions in place, enforcing a concept to only consist of instances or only other concepts.

A concept can be subsumed by another concept or it can subsume another concept itself. This property is useful, because of this it is possible to create a hierarchy within the concepts. For example, “building” subsumes “floor” because everything that is a member of the floor concept must also be part of the building concept. Partitions are used to determine where an instance goes within the ontology. A partition is a set of concepts and rules, if these rules determine that an instance is places in one concept alone it is a disjoint partition. If the rules ensure that every object in the upper class is an instance of at least one of the partition classes the partition is an exhaustive partition.

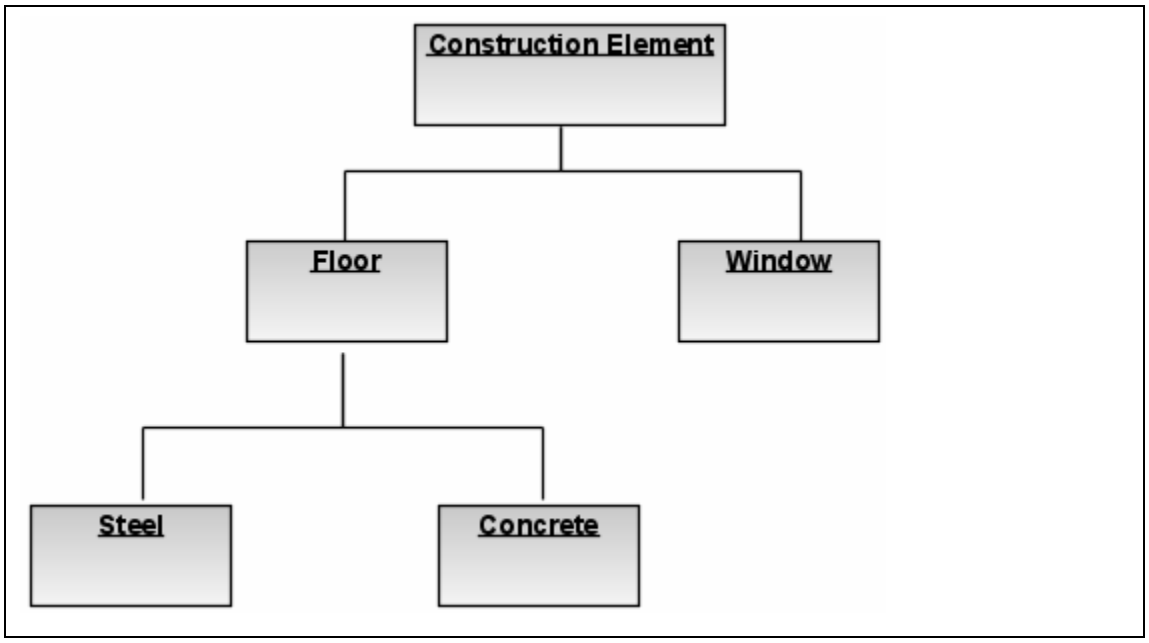


Figure 1 – Concept subsuming

This example is a part of a construction ontology, the floor concept is a construction element and concrete is part of the floor. Because of the hierarchy, concrete is also a construction element.

3.3. Attributes

Every instance in an ontology can have attributes assigned to it. These attributes describe what the instance is, specific information about the instance is captured in the ontology in this way. The attributes must have a name and a value, this value can be a simple type (a string or integer) but also a complex type. Because of this, a single attribute can have multiple values captured in a list for example. A simple example from the construction ontology might be a specific type of window.

```
Name = Special Insulating Glass
Tint = light blue
Number of panes = {3,4,5}
Width = 3200 mm
Height = 1800 mm
```

Example 1 – Attributes example

This type of glass, Special Insulating Glass, has a light blue tint with a fixed size of 3200 by 1800 millimetres. It is available with 3, 4 and 5 panes for added insulation. Because of these attributes within an ontology, reasoning applications can find the right type of window for every application.

3.4. Relations

The attributes of instances are also used to describe relationships between instances. The most used relationship is an attribute with another instance as its value. To stay with the window example, one attribute might be “Next size = Large Special Insulating Glass”, pointing to another instance with that name. That instance should be a window with a bigger size than the current one. By using this kind of references to other attributes, the semantics of the domain are being described. One giant web of related instances is the final product.
The subsumption as described in paragraph 3.2 is also a very important part of the relationships between instances. This defines which instances are part of concepts and where they are located in the ontology.

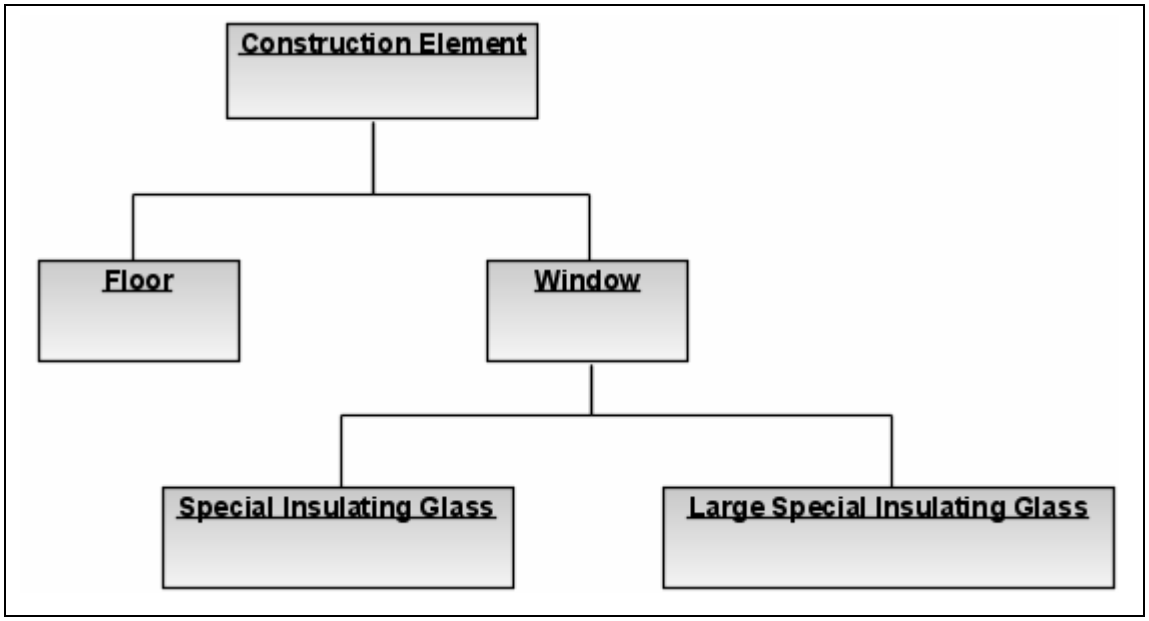


Figure 2 – Subsumption

Both the Special Insulating Glass instance is a window and the Large Special Insulating Glass is a window, which in turn is a construction element. Since we created this small example, it is not hard to see the relationships. In a large ontology, there are too many relationships to oversee. Besides this child-parent relationship, ontologies can contain another kind of relationship. This is the meronymy relation, instances can be part of another instance while together they form a new composite instance. For example, insulating rubber can be part of the Special Insulation Glass. Domain-specific relationships are also common within ontologies but these are, hence the name, domain-specific. It is impossible to describe all these type of relationships. It suffices to say that all these relationships are there to further capture the semantics of the ontology. In our example, we could add that the glass from the Special Insulation Glass type window is created in Rotterdam, this is in turn part of the Randstad which is located in the Netherlands. These kinds of relationships can go on almost indefinitely. Because of this it is possible for reasoning software to answer a question like ‘which windows are composed with glass from the Netherlands, wider then 3000 millimetres’.

4. RDF and RDF Schemas

This paragraph will discuss the basis on RDF and RDF Schemas. To fully grasp all concepts discussed in the following part, we advise the reader to have read the previous chapters, or to be familiar with XML and XBRL.

4.1. RDF

As RDF is shorthand for “Resource Description Framework” it is built on the fundamentals of XML and tries to provide a generic framework to describe various resources on the web.
To accomplish this, a RDF-file consists of one or more “resources”, which are given between the `<rdf:RDF>` and `</rdf:RDF>` tags.

A RDF-document was not intended to be read by humans, it tries to enable computers and programs to read and truly understand what the contents of the file are by supplying some additional metadata together with the resource itself.

An example RDF-file is like the following:

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
.
. Resources go here
.
</rdf:RDF>
```

Example 2 – The RDF document definition

The underlying structure of any expression in RDF is a collection of triples, each consisting of a subject, a predicate and an object. A set of such triples is called an RDF graph. This can be illustrated by a node and directed-arc diagram, in which each triple is represented as a node-arc-node link (hence the term "graph").

Figure 3 – The RDF relationships

Each triple represents a statement of a relationship between the things denoted by the nodes that it links. Each triple has three parts:
A subject, an object, and a predicate (also called a property) that denotes a relationship.

The direction of the arc is significant: it always points toward the object.

The nodes of an RDF graph are its subjects and objects.

The assertion of an RDF triple says that some relationship, indicated by the predicate, holds between the things denoted by subject and object of the triple. The assertion of an RDF graph amounts to asserting all the triples in it, so the meaning of an RDF graph is the conjunction (logical AND) of the statements corresponding to all the triples it contains.

4.1.1. Resources

A resource in a RDF-document is defined by the use of the tag `<rdf:Description>`.
Every resource consists of a namespace in the rdf:about attribute and zero or more other attributes and properties. Every attribute has its attribute-value, and every property has its property-value.
Reconsidering the XML-document we mentioned earlier, the DVD example can be reused to be displayed in RDF.
However, as RDF focuses more on several items, we will add another DVD.

This example uses only elements as properties of the DVD, the properties can also be given as attributes of the `<rdf:Description>`-tag. The latter is not recommended as of the lesser ease of navigatability.

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:dvd="http://www.dvdshop.edu/dvd#">

<rdf:Description rdf:about="http://www.dvdshop.edu/dvd/Lock, Stock and Two Smoking
Barrels">
  <dvd:director>Guy Ritchy</dvd:director>
  <dvd:country>USA</dvd:country>
  <dvd:language>English</dvd:language>
  <dvd:price>10.99</dvd:price>
  <dvd:year>1998</dvd:year>
</rdf:Description>

<rdf:Description rdf:about=" http://www.dvdshop.edu/dvd/Le Transporteur II">
  <dvd:director>Louis Leterrier</dvd:director>
  <dvd:country>France</dvd:country>
  <dvd:language>English</dvd:language>
  <dvd:price>14.99</dvd:price>
  <dvd:year>2005</dvd:year>
</rdf:Description>

</rdf:RDF>
```

Example 3 – The DVD's in RDF

The extensibility is greatly enhanced by the next feature: a resources property can also be a resource. This way, a property can point to another resource (somewhere else) on the web, and be described more precisely, or described evenly throughout all resources.

```
<rdf:Description rdf:about="http://www.dvdshop.edu/dvd/Lock, Stock and Two Smoking
Barrels">
  <dvd:director rdf:resource="http://www.dvdshop.edu/dvd/Guy Ritchy" />
  <…>
</rdf:Description>
```

Example 4 – A resource as property

4.1.2. Containers

When a property has more that one possible value, for instance the genres of a DVD, they cannot be given the same way all other property-values are given, as a property has to be unique within a resource.

To overcome this problem, a container has been introduced. A container is simply a small collection of values which can be regarded of as either ordered or unordered. Three different types of containers have been defined in the RDF-definition: Bag, Seq and Alt.

The following table lists the aspects of the container-types.

	<rdf:Bag>	<rdf:Seq>	<rdf:Alt>
Ordered	No	Yes	No
Duplicates	Yes	Yes	No

Table 1 – Container-types

Note: a container only lists the elements that are there, it doesn't list the elements not allowed or the element that are not part of the container. This is, like XML, all done in the attached schema.

4.2. RDF Schema

Unlike XML Schema where the schema is used to present the data in a readable fashion to humans, RDF Schema intends to show data in a knowledge presentation. RDF Schema provides basis elements for the description of ontologies (a being described in the previous chapter), otherwise called RDF vocabularies.

While an XML Schema is only a derivative of XML, a RDF Schema is more closely bound to RDF itself and uses several of RDF's main building blocks.

4.2.1. Classes and subclasses

In a RDF Schema groups of different resources may be separated and identified by the declaration of a class of resources. Members of a class are known as instances, which is not to be swapped with the term instance from XML.

As an example of an instance, please think back of the glass panes mentioned in paragraph 3. Special Insulating Glass and Large Special Insulating Glass are both instances of the class Window; but to make things more complicated they are also children or subclasses of the class Window.

The actual window used in construction of the house is the instance of the class Window. What type it is, is not relevant in this case.

RDF distinguishes between a class and the set of its instances. Associated with each class is a set, called the class extension of the class, which is the set of the instances of the class. Two classes may have the same set of instances but be different classes.
A class may be a member of its own class extension and may be an instance of itself.

A class is identified with the <rdfs:Class>-tag, and a subclass is depicted with the <rdfs:subClassOf>-tag. All resources being an instance of the subclass C, which has C' as it's parent, are also instances of C'; yet this doesn't hold the other way around with one exception: if a class C' is a super-class of a class C, then all instances of C are also instances of C'. The latter is very uncommon though.

All things described by RDF are called resources, and are instances of the class rdfs:Resource. This is the class of everything. All other classes are subclasses of this class. rdfs:Resource is an instance of rdfs:Class.

Two other types have also been defined:

- rdfs:Literal is the is the class of literal values such as strings and integers.
- rdfs:Datatype is both an instance of and a subclass of rdfs:Class. Each instance of rdfs:Datatype is a subclass of rdfs:Literal.

4.2.2. Domain and range

Two tags are defined to represent the domain and range of resources in an RDF Schema, rdfs:domain and rdfs:range. rdfs:domain of an rdf:property declares the class of the subject in a triple using this property as predicate.
rdfs:range of an rdf:property declares the class or datatype of the object in a triple using this property as predicate.

5. OWL

5.1. What is OWL?

As mentioned earlier in this paper, Ontologies are used to capture knowledge about some domain of interest. The Web Ontology Language, atypically abbreviated to OWL, is a language to represent such Ontologies. OWL extends existing web standards like XML (see the previous paragraph I), RDF and RDF Schema (see previous parts) and is represented into three species. The first being the OWL Lite, second OWL DL an extension of OWL Lite and last OWL Full a full union of RDF in the OWL syntax.

5.2. Build-up of an OWL Ontology

An OWL Ontology consist of Individuals, Properties, and Classes. This terminology can be matched with that as we previous mentioned in the paragraph about Ontologies.

5.2.1. Individuals

Individuals can be matched with the Instance element as explained in paragraph 3. Individuals are in OWL the representation of elements that we, as builders and/or users of the OWL Ontology, are interested in. In OWL there is a possibility to give different names to the same individual. In the DVD example the individual of the DVD “Le Transporteur II” can have the name “Le Transporteur II” as well as “Le Transporteur 2” linked to it. In this situation there must be explicit stated that individuals are the same as each other, this is obligated in OWL as is the situation where individuals are different to each other.

5.2.2. Properties

Properties are the relations between two individuals. An property “hasDirector” might link the individual “Lock, Stock and Two Smoking Barrels” with the individual “Guy Ritchie”, or a property spokenLanguage might link the individual “Le Transporteur II” with the individual “English”. Properties can have inverse appearance, so can the individual “Guy Ritchie” also be linked to “Lock, Stock and Two Smoking Barrels” with the property “hasDirected”. The properties element in OWL can be linked with that of the relations as explained in paragraph 3.

5.2.3. Classes

In OWL Classes, comparable with the Concepts element in the Ontologies Chapter, sets of individuals are represented. These representations are given in formal mathematical descriptions. In the OWL-DL representing form of OWL, super-sub class relationships can be computed automatically by a reasoner when a correct form of Description Language, hence the name OWL-DL, is being used. So can be presented that all movies made by the individual “Guy Ritchie” are in “English”.

5.3. OWL RDFS Syntax

For the representation of OWL RDF Schemas as explained in paragraph 4 can be used. In the following example presented will be the class link as mentioned previous in which is linked that all movies that are made by Guy Ritchy are spoken in English.

```
<owl:Class>
  <owl:intersectionOf rdf:parseType="collection">
        <owl:Class rdf:about="#Movie"/>
        <owl:Restriction>
              <owl:onProperty rdf:resource="#hasDirector"/>
              <owl:toClass>
                 <owl:unionOf rdf:parseType="collection">
                       <owl:Class rdf:about="#Guy Ritchie"/>
                       <owl:Restriction>
```

Example 5 – Part 1 of a RDFS

Here in the first part is mentioned that a restriction is made on all the movies that have as Director Guy Ritchie.

```
                          <owl:onProperty rdf:resource="#spokenLanguage"/>
                          <owl:hasClass rdf:resource="#English"/>
```

Example 6 – Part 2 of a RDFS

The Restriction is that the spoken language in the movie must be English.

```
                       </owl:Restriction>
                 </owl:unionOf>
              </owl:toClass>
        </owl:Restriction>
  </owl:intersectionOf>
</owl:Class>
```

Example 7 – Part 3 of a RDFS

In the OWL different constructs are available for use. A summary on those constructs is presented by the W3C on their website (http://www.w3.org/TR/owl-features/).

VI. Transforming XBRL into OWL: towards a useful Ontology

1. Introduction

As was already concluded in the previous chapters the NTP taxonomies and how XBRL has been used in general has some important disadvantages: the taxonomies have become very complex in nature; layer upon layer is built by including, extended, generalizing and more of the like. It is not possible for a human anymore to get a grasp what a single taxonomy encompasses and how it is structured, since it most of the time consists of thousands of elements, distributed among several tens of files.

Since ontologies are a way to represent information and their relationships in a human understandable way and at the same time let computers also understand it (see chapter V.3), an ontology knowledge base containing business reporting information may be very useful.

This paragraph elaborates on our attempt to transform XBRL into a useful ontology which can contain information on organizations typically found in external business reports.

The next paragraph describes the goals and boundaries we had in mind for constructing the ontology. Then the ontology is presented, followed by a description on its construction process and some detailed decisions that were made. After that some issues and open questions are described, followed by a description of the tools we used for creating the ontology and how we used them.

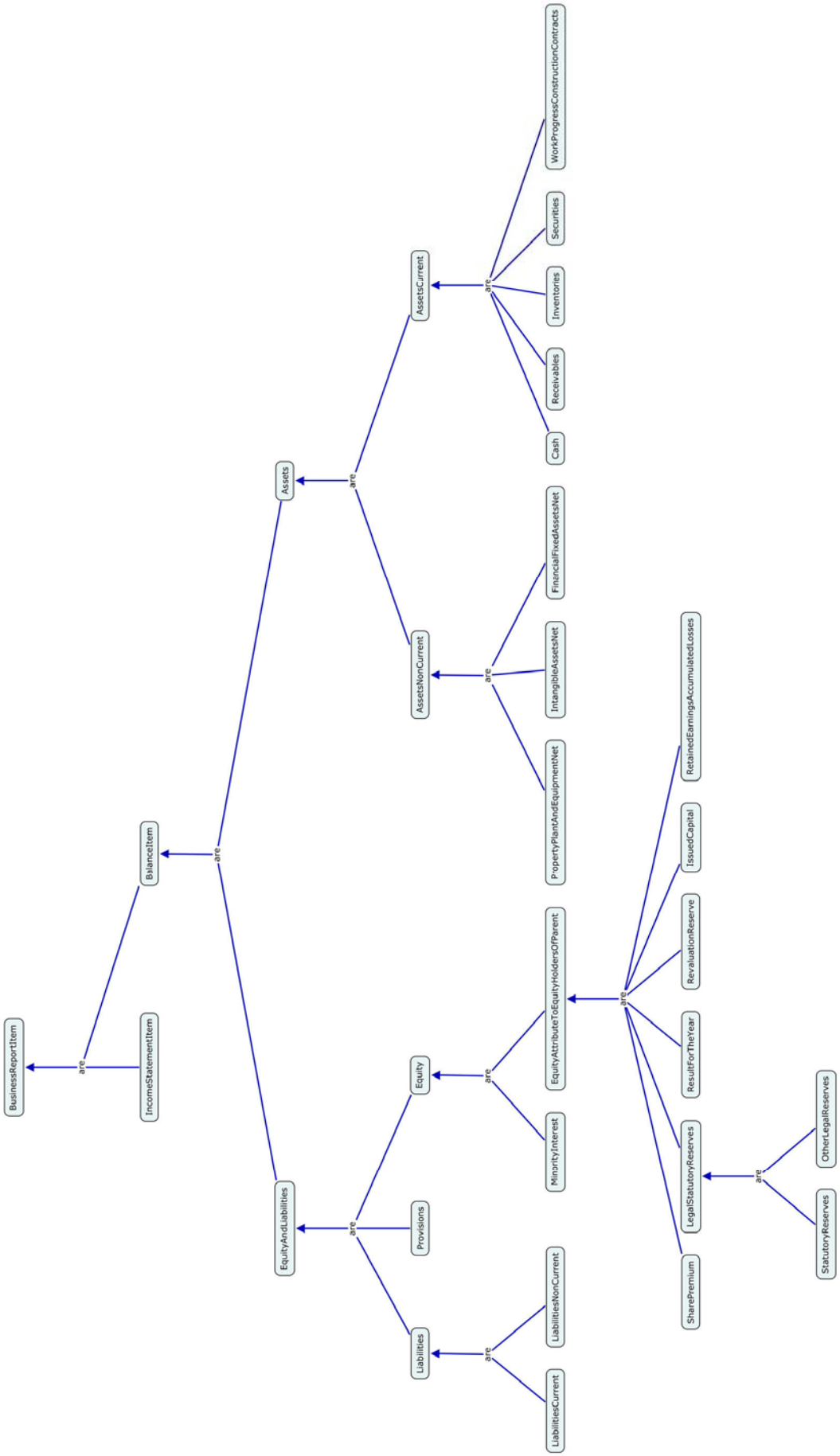
2. Goals and boundaries

When constructing the ontology the following goals and boundaries were kept in mind:

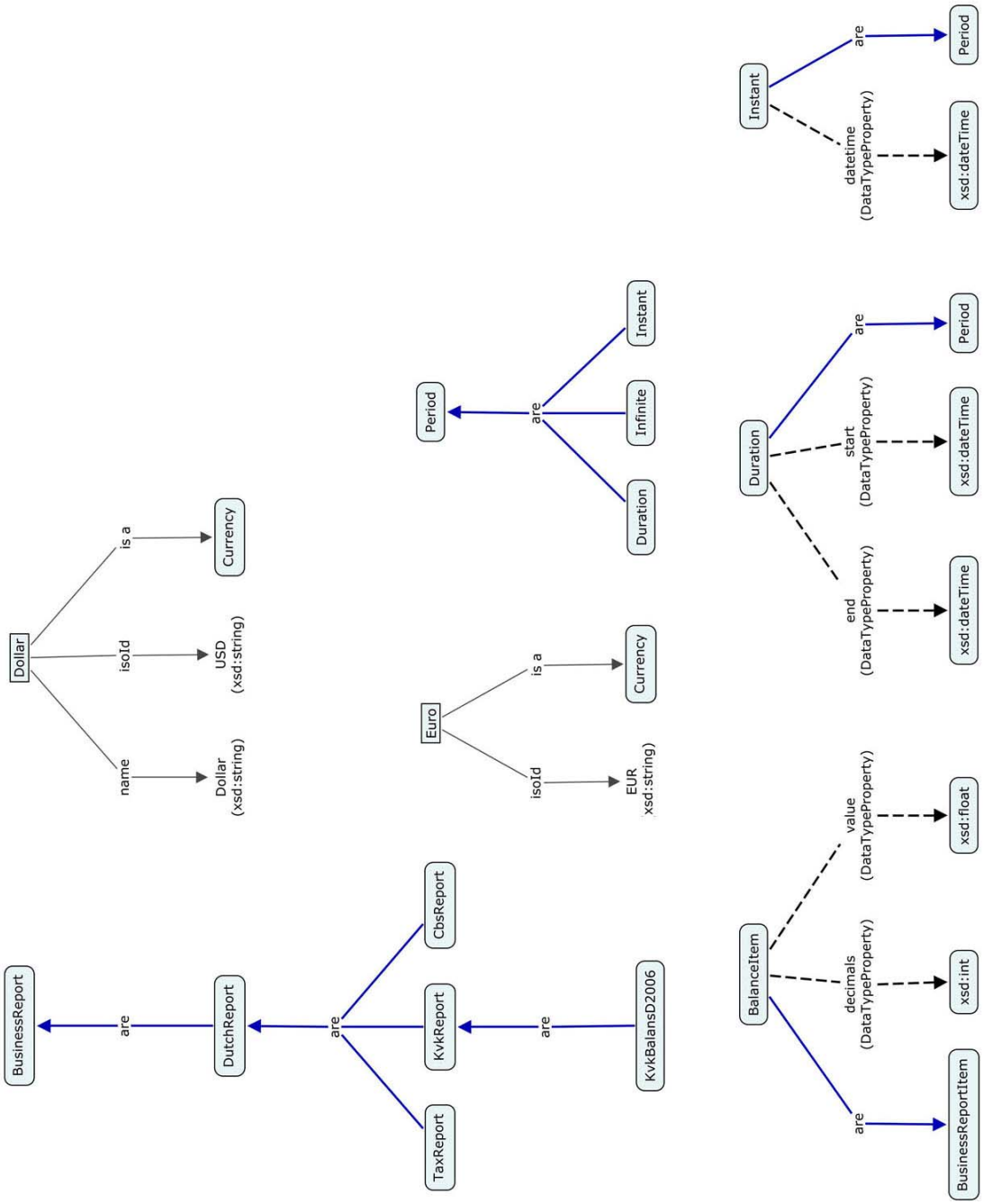
- don't make the ontology overly complex and generic
- don't forget to think of an application of the ontology
- started off easy and build up from there
- for an Ontology-based knowledge base to be successful, it should be reasonably simple, useful and extendible

3. The ontology and how it is constructed

The following pages contain the various parts of our ontology. The OWL-file containing the ontology can be obtained from the authors.


BusinessReportItem
are
IncomeStatementItem
BalanceItem
are
Assets
are
AssetsCurrent
are
WorkProgressConstructionContracts
Securities
Inventories
Receivables
Cash
AssetsNonCurrent
are
FinancialFixedAssetsNet
IntangibleAssetsNet
PropertyPlantAndEquipmentNet
EquityAndLiabilities
are
Equity
are
EquityAttributeToEquityHoldersOfParent
are
RetainedEarningsAccumulatedLosses
IssuedCapital
RevaluationReserve
ResultForTheYear
LegalStatutoryReserves
are
OtherLegalReserves
StatutoryReserves
SharePremium
MinorityInterest
Provisions
Liabilities
are
LiabilitiesNonCurrent
LiabilitiesCurrent

KvkBalansD2006
_period
(>>=1)
Period
_retainedEarningsAccumulatedLosses
(>>=1)
RetainedEarningsAccumulatedLosses
_otherLegalReserves
(>>=1)
OtherLegalReserves
_intangibleAssetsNet
(>>=1)
IntangibleAssetsNet
_securities
(>>=1)
Securities
_kvkReport
(>>=1)
KvkReport
_equityAttributeToEquityHoldersOfParent
(>>=1)
EquityAttributeToEquityHoldersOfParent
_minorityInterest
(>>=1)
MinorityInterest
_financialFixedAssetsNet
(>>=1)
FinancialFixedAssetsNet
_liabilitiesCurrent
(>>=1)
LiabilitiesCurrent
_issuedCapital
(>>=1)
IssuedCapital
_currency
(>>=1)
Currency
isoId
(DataTypeProperty)
xsd:string
_receivables
(>>=1)
Receivables
_assets
(>>=1)
Assets
_inventories
(>>=1)
Inventories
_workProgressConstruct
(>>=1)
WorkProgressConstructi
KvkBalansD2006
_workProgressConstructionContracts
(>>=1)
WorkProgressConstructionContracts
_provisions
(>>=1)
Provisions
_assetsCurrent
(>>=1)
AssetsCurrent
_statutoryReserves
(>>=1)
StatutoryReserves
_sharePremium
(>>=1)
SharePremium
_liabilities
(>>=1)
Liabilities
_equityAndLiabilities
(>>=1)
EquityAndLiabilities
_propertyPlantAndEquipmentNet
(>>=1)
PropertyPlantAndEquipmentNet
_cash
(>>=1)
Cash
_legalStatutoryReserves
(>>=1)
LegalStatutoryReserves
_assetsNonCurrent
(>>=1)
AssetsNonCurrent
_liabilitiesNonCurrent
(>>=1)
LiabilitiesNonCurrent
_equity
(>>=1)
Equity
_revaluationReserve
(>>=1)
RevaluationReserve
_resultsForTheYear
(>>=1)
ResultForTheYear


BusinessReport
are
DutchReport
are
CbsReport
KvkReport
TaxReport
are
KvkBalansD2006
Dollar
is a
Currency
isoId
USD
(xsd:string)
name
Dollar
(xsd:string)
Euro
is a
Currency
isoId
EUR
xsd:string)
Period
are
Instant
Infinite
Duration
Instant
are
Period
datetime
(DataTypeProperty)
xsd:dateTime
Duration
are
Period
start
(DataTypeProperty)
xsd:dateTime
end
(DataTypeProperty)
xsd:dateTime
BalanceItem
value
(DataTypeProperty)
xsd:float
decimals
(DataTypeProperty)
xsd:int
are
BusinessReportItem

The process for constructing this ontology and the decisions made along the way are:

- We started off with a very simple taxonomy: the Digiforce example and translated it to an ontology by hand.
- This Ontology we then discussed and extended with more concepts and also some instances.
- Since we wanted to work towards a usable, simple but extendable ontology we decided that taking a simple taxonomy from the NTP ([Nederlandse Taxonomie Project], 2006 Oct 17) was a good idea: at this moment the way to get information on companies from the KVK is to purchase separate business reports from the website. What if all reports were loaded into one knowledge base: would this be helpful? Would it lead to a useful base of information, extracting new information previously unseen? Without specific applications in mind, we thought that it probably would.
- The taxonomy of choice has become "Balans D 2006" from the KVK (Chamber of Commerce) reports, which contains amongst others a simple presentation linkbase for a balance sheet. The balance sheet consists of 35 elements.
- The items which should be included in the report are all items specified on a more general level in included taxonomies, including IFRS and Dutch legislation. The report structure itself is specified as a presentation linkbase in the report taxonomy. This let us to reason that generally speaking all item specifications which could be included in a report are specified on a more generic level, overlapping various reports and that the report itself is just a set of these items demanded by a certain party.
- This let to a split of the ontology in two main parts: a generic part containing balance item specifications in a parent-child hierachy and a specific part containing a class specifying which items should be present for a report.
- For the generic part a class for each element was created (Assets, Non-current Assets, Liabilities, Equity etc.), changing their name to a more meaningfull one if necessary.
- The parent-child hierarchy is comparable to the presentation link structure in the taxonomy. Extra levels of classes were added, if this would give more information to a user.
- To keep the ontology simple and logical, the complex item types were not created (such as MonetaryItemType), since this will confuse the user. Instead simple datatypes are used to contain values or indicate how many decimals are used. This does produce some redundancy, but will make the ontology much more readible.
- The specific part currently contains only one report: KvkBalansD2006 which contains all the items which should be present according to the presentation linkbase of the chosen NTP taxonomy. Some parent classes are added to identify how additional report classes can be added in the future.

Besides the general process description above, some detailed technical choices we made in the ontology:

- From the taxonomy it became appearant that Equity and Liabilities can contain three types of items: Equity, Liabilities and Provisions. For this reason three child classes were made, while nog changing the name of the parent class. The latter was done since the term “Equity and Liabilities” is generally accepted for that item.
- Total monetary values are stored in the knowledge base, for example: the value of an Assets instance should be the total of the value of the instances of AssetsNonCurrent and AssetsCurrent for a specific report instance.
- Plural class names are used as is customary in business reports, so “Assets” instead of “Asset”, since an instance of the class will contain the value of all assets, not just one.
- The “context” item from XBRL is not used. Instead an instance of a child type of the Period class should be related to a report instance. The same holds for currency: a Currency instance (Dollar, Euro) should be related to a report instance to indicate the currency the report is in.
- It is possible to define a complete parent-child hierarchy of all possible items in a business report by defining all parts as “items” from the top down. If this was not done, it would not be possible to add arbitrary report classes in the future without rendering generic classes incomplete or obsolete.

4. Issues and open questions

The next step for developing the ontology would be to add numerous instance reports in order to create a knowledge base. This knowledge base should then be queried and explored to determine if it is possible to extract new and interesting information which could not be retrieved previously from existing databases and/or applications. This exploration will probably result in new insights into the knowledge base's uses, possibilities and shortcomings, leading to development of new versions of the ontology. Over time the ontology and knowledge base can then lead to a new source of information.

During the construction of the ontology the following issues and open questions were raised, which should be addressed in the future:

- It was reasoned that a tool using the ontology would be sophisticated enough to determine the structure of a report instance (instance of KvkBalansD2006) from the parent-child hierarchy of its items. But will this be the case? If not, then a different class structure should be made.
- Should a class property be added to the generic classes to indicate their parent instance? This question relates to the previous one: it was reasoned that a tool would be able to group all instances belonging to a report based on the fact that they are related to a single report instance. If this is not the case, then adding a parent property may be a solution.

- Does a generic structure of items hold when the ontology is extended for use by other reports and parties? Or does this create inconsistencies? These are important questions for the general use, possible usage and continued simplicity of the ontology.

5. Tools used for constructing the ontology

In order to display the XBRL files in a more understandable way, we used two different tools. The Semansys Taxonomy viewer is a free tool, it can read taxonomy files and display them in a tree view of list view. This tool takes all referenced linkbases into account and processes them to display the information captured among different files in one, understandable overview. To view and possibly edit these XBRL files, we also used another tool made by Fujitsu. The Fujitsu XBRL tools are two different programs, the Taxonomy Editor and the Instance Creator. These tools also present the XBRL files in a more understandable way for people to read.

To create OWL files, we used different tools. The main tools used are Protégé and IHMC Cmap Tools. The first, protégé, is a major tool used by a lot of people and is widely accepted as producing correct OWL files. This tool however lacks the possibility to display and edit ontologies in a graphical way. This is where Cmap comes into play, the version we are using is 4.08 COE as this is the last version to support OWL. The newest version does not have any support for OWL files. Cmap Tools gives users the possibility to view and edit ontologies in a graphical way, this is more intuitive than the lists provided by protégé. After saving the ontology as an OWL file it can be imported by protégé to check if the produced file is correct.

VII. List of images

VIII. List of examples and tables

IX. References

Arciniegas, F. A. (2000 Sep 18), “What Is XLink”, <http://www.xml.com/pub/a/2000/09/xlink/index.html>. Accessed 2007 Jan 20.

Berners-Lee, T., Hendler, J. and Lassila, O. (2001 May), “The Semantic Web”, <http://www.sciam.com/article.cfm?articleID=00048144-10D2-1C70-84A9809EC588EF21>. Accessed 2007 Feb 03.

Burg, H. J. van, (2005), ‘Reducing administrative burdens through standardisation’, <http://www.xbrl-ntp.nl/english/Overviewntp10.pdf>

[Colostate.edu] (2007-01-27), “Sample Taxonomy Schema”, http://ls103024.csulinuxhub.colostate.edu/moit04/xbrl/V2AnnotatedSchema.htm. Accessed 2007-01-27.

[Ernst&Young] (2007-01-26), “What is a taxonomy?”, http://www.ey.com/global/content.nsf/International/XBRL-What_are_Taxonomies. Accessed 2007-01-26.

[Fujitsu] (2007 Feb 05), “Fujitsu XBRL Tools”, <http://software.fujitsu.com/en/interstage-xwand/activity/xbrltools/> Accessed 2007 Feb 05.

Horridge, M., Knublauch, H., Rector, A., Stevens, R., Wroe, C. (2004 Aug 27), “A Practical Guide To Building OWL Ontologies Using The Protége-OWL Plugin and CO-ODE Tools Edition 1.0”, <http://www.co-ode.org/resources/tutorials/ProtegeOWLTutorial.pdf>. Accessed 2007 Feb 05.

[IHMC.us] (2007 Feb 05), “IHMC CmapTools v4.08 COE”, <http://cmap.ihmc.us/coe/beta/install.htm> Accessed 2007 Feb 05.

[man.ac.uk] (2007 Feb 05), “Tutorial on OWL”, <http://www.cs.man.ac.uk/~horrocks/ISWC2003/Tutorial/>. Accessed 2007 Feb 05.

[Semansys] (2007-01-30), “Taxonomy Viewer V2”, http://www.semansys.com/taxonomy_viewer.html. Accessed 2007-01-30.

[Stanford.edu] (2007 Feb 05), “What is an ontology and why we need it”, <http://protege.stanford.edu/publications/ontology_development/ontology101-noy-mcguinness.html>. Accessed 2007 Feb 05.

[Stanford.edu] (2007 Feb 05), “Protégé, the free, open source ontology editor and knowledge-base framework”, <http://protege.stanford.edu/> Accessed 2007 Feb 05.

Vreeburg, T. e.a., “Web Enabled Business Reporting, de invloed van XBRL op het verslaggevingsproces”, Ernst & Young, Kluwer, 2004.

Walmsley, P. (2002 Jan 18), “XML Schema: An Overview”, <http://www.informit.com/articles/article.asp?p=25002&seqNum=7&rl=1>. Accessed 2007 Jan 20.

[W3C] (2006 Aug 16). “Namespaces in XML 1.0 (Second Edition)”, <http://www.w3.org/TR/REC-xml-names/>. Accessed 2007 Jan 20.

[W3C] (2004 Feb 10), “RDF Vocabulary Description Language 1.0: RDF Schema”, <http://www.w3.org/TR/rdf-schema/>. Accessed 2007 Feb 05.

[W3C] (2004 Feb 10), “RDF/XML Syntax Specification (Revised)”, <http://www.w3.org/TR/2004/REC-rdf-syntax-grammar-20040210/>. Accessed 2007 Feb 05.

[W3Schools] (2007 Jan 22), “XML Attributes”, <http://www.w3schools.com/xml/xml_attributes.asp>. Accessed 2007 Jan 22.

[W3C] (2001 Jun 27), “XML Linking Language (XLink) Version 1.0”, <http://www.w3.org/TR/xlink/>. Accessed 2007 Jan 20.

[W3Schools] (2007 Jan 20), “XML Namespaces”, <http://www.w3schools.com/xml/xml_namespaces.asp>. Accessed 2007 Jan 20.

[W3Schools] (2007 Jan 20), “XML Schema Example”, <http://www.w3schools.com/schema/schema_example.asp>. Accessed 2007 Jan 20

[W3Schools] (2007 Jan 20), “XML Schema Import Element”, <http://www.w3schools.com/schema/el_import.asp>. Accessed 2007 Jan 20.

[W3Schools] (2007 Mar 07), “XML Schema Restrictions/Facets”, <http://www.w3schools.com/schema/schema_facets.asp>. Accessed 2007 Mar 07.

[W3Schools] (2007 Jan 20), “XML Schema Tutorial”, <http://www.w3schools.com/schema/default.asp>. Accessed 2007 Jan 20.

[W3Schools] (2007 Jan 20), “XML Syntax”, <http://www.w3schools.com/xml/xml_syntax.asp>. Accessed 2007 Jan 20.

[Wikipedia] (2007 Jan 22), “Extensible Stylesheet Language”, <http://en.wikipedia.org/wiki/Extensible_Stylesheet_Language>. Accessed 2007 Jan 22.

[Wikipedia] (2007 Feb 05), “Ontology (Computer Science)”, <http://en.wikipedia.org/wiki/Ontology_%28computer_science%29>. Accessed 2007 Feb 05.

[Wikipedia] (2007 Feb 05), “Semantic Web”, <http://en.wikipedia.org/wiki/Semantic_Web>. Accessed 2007 Feb 05.

[Wikipedia] (2007 Feb 05), “Web Ontology Language”, <http://en.wikipedia.org/wiki/Web_Ontology_Language>. Accessed 2007 Feb 05.

[Wikipedia] (2007 Jan 22), “XML Schema (W3C)”, <http://en.wikipedia.org/wiki/XML_Schema>. Accessed 2007 Jan 22.

[Wikipedia] (2007 Jan 22), “XSL Transformations”, <http://en.wikipedia.org/wiki/XSL_Transformations>. Accessed 2007 Jan 22.

[XBRL.org] (2007-01-26), “XBRL Taxonomies”, http://www.xbrl.org/Taxonomies/. Accessed 2007-01-25.

[XBRL.org] (2006-12-18), “XBRL 2.1 Recommendation”, http://www.xbrl.org/Specification/XBRL-RECOMMENDATION-2003-12-31+Corrected-Errata-2006-12-18.htm#_Toc156209131. Accessed 2007-01-25.

[XML.com] (1998 Oct 3), “A Technical Introduction to XML”, <http://www.xml.com/pub/a/98/10/guide0.html>. Accessed 2007 Jan 20.

[ZVON] (2007 Jan 20), “9. Attribute arcrole”, <http://www.zvon.org/xxl/xlink/Output/xlink_refs.html>. Accessed 2007 Jan 20.