Translating XBRL Into Description Logic. An Approach Using Protégé, Sesame & OWL

Thierry Declerck and Hans-Ulrich Krieger

German Research Center for Artificial Intelligence (DFKI)
Stuhlsatzenhausweg 3, D-66123 Saarbrücken, Germany
{declerck,krieger}@dfki.de

Abstract. In the context of the eTen project, WINS, a web-based business intelligence service to public and private financial institutions has been designed and implemented. One of the goals of the project was to provide new financial knowledge on companies from information gathered through interoperable information services. The services were implemented under the new emerging standard XBRL used for financial reporting. We sketch how relevant financial information was extracted from annual financial reportings. We also show at the same time the limitations we encountererd with the XBRL schema, due to the lack of reasoning support over XML-based data and information extracted from documents. To overcome these difficulties, we describe the "ontologization" of XBRL, which we assume to be a necessary requisite for large intelligent web-based financial information and decision support systems.

1 General Background

In the context of the eTen project, WINS, a web-based business intelligence service to public and private financial institutions has been designed and implemented.[1] One of the goals of the project was to provide new financial knowledge on companies from information gathered through interoperable information services [1]. The services were implemented under the new emerging standard called XBRL (eXtensible Business Reporting Language) used for financial reporting. In the following, we sketch how relevant financial information was extracted from annual financial reportings. We also show at the same time the limitations we encountererd with the XBRL schema, due to the lack of reasoning support over XML-based data and information extracted from documents that is finally mapped onto XBRL instances. In the first part of our submission, we just summarize our way of information extraction guided by XBRL, and afterwards describing our work dedicated to the *ontologization* of XBRL, which we assume to be a necessary requisite for large intelligent web-based financial information and decision support systems.[2] The work described here will be further carried out within an Integrated Project of the 6th Framework, called MUSING (MUlti-Industry, Semantic-based next generation business Intelligence).

[1] ETEN 2003/1, Grant agreement nr. C51083. WINS stands for Web-based Intelligence for common-interest fiscal Networked Services'.
[2] For more information, see XBRL International: http://www.xbrl.org.

2 Incremental Information Extraction Guided by XBRL

The next three subsections present the basic setting, viz., XBRL-guided information extraction of structured and unstructured documents.

2.1 Knowledge-Driven Information Extraction from Structured and Unstructured Documents

The actual input for the information extraction (IE) task in WINS consists of balance sheets in PDF format, containing structured forms (tables) and free text (included, for example, in the annexes of balance sheets). Relevant information extracted from these sources are merged and mapped onto the XBRL format.

A terminological clarification should be given at this place. IE often refers to the task of filling user- or application-defined templates with the result of information detected by natural language analysis tools in textual documents. For certain applications, knowledge bases are available, supporting the IE task. Such knowledge might consist of taxonomies, thesauri, or ontologies. In this case, knowledge-driven IE tries to *populate* knowledge bases with *instances* detected in the textual documents. This was the situation in WINS, where the XBRL taxonomy was guiding the IE task through the analysis of both tabular data and free text. In the end, an XBRL structure should be instantiated with the information extracted from the annual reports of companies.

2.2 The Mapping Process from Text to XBRL

The mapping process has been implemented within a Web service made available to the WINS partners. The Web service operates on PDF/text files from WINS data providers and returns files, containing the data in XBRL format. In a first step, text and tables from the PDF documents were extracted. It was also necessary to apporimatively recontructs the original layout, which is getting lost in the PDF-to-text conversion.

Once this has been done, the WINS information extraction module inspects the generated HTML documents, trying to find correspondences in the text of the tables for labels of concepts contained in the overall XBRL taxonomy. But not only the detection of realizations of XBRL concepts in the document is important. The extraction tools must also detect relevant dates in the tables as well as currencies used, so that the figures contained in the tables , e.g., balance and profit & loss (P&L) tables, are getting their correct interpretation.

Since the XBRL taxonomy is also considering information about a company as such (name, address, number of employees, etc), the extraction tools need to detect this kind of information. In order to obtain such information, we implemented a simple named entity recognition algorithm for detecting names of companies, locations, and relevant persons.

2.3 Incremental Process

Even though the automatic process of mapping structured data into XBRL is not perfectly accurate, it already brings a significant efficiency improvement in providing XBRL data for various applications, such as self-assessment-based company rating.

But more is necessary. *Aggregation* of information is needed, consisting, for example, in adding to the XBRL document information that is not included in the tables. As a first step, annexes present in the annual report of a company are analyzed and the relevant information for the XBRL document is extracted. This is the place where linguistic analysis and text mining comes into play, since the information is no longer available in structured form (positions in a table), but given as free text.

Let us give an example of incremental processing from an annual report. In a P&L table, a position is reserved for *Umsatzerlöse* (trading profit) with a sum of 159,356 K Euro. The PDF-to-XBRL converter detects this fact and generates the corresponding XBRL code. But the table contains a reference to a section in the annex, and here, we find in free text that parts of the *Umsatzerlöse* has been reached abroad:

> *Von den Umsatzerlösen wurden 78.299 Tsd. Euro (Vorjahr 1.653 Tsd. Euro) im Ausland erzielt.*

This is a case, where obviously text analysis is needed if one wants to extract this information, since a simple pattern wouldn't suffice for ensuring the extraction task. The knowledge base (here the XBRL taxonomy), tells us that *Umsatzerlöse* in this portion of text is a relevant term. Hence, this information is extracted and combined with the information gained from the table. Some more facts are also needed, which can be considered as *soft* facts; for instance, changes in the management structure of a company, analysis of trend in a special branch, information that is present in documents external to the annual report, etc. This information will be mapped onto XBRL, if there are some terms in the taxonomy corresponding to this information. Other relevant information needs to be encoded in a usable way, so that additional relations betwen soft and hard facts can be stated or inferred. In this case the incremental building of XBRL-encoded information is not trivial at all, since documents need to be concerned that have been written during a longer period of time. There is a need here for a more principled way of performing merging of information, based on temporal reasoning (see, e.g., [2]). Therefore, we opt for a *porting* of XBRL to a representation language that supports the detection of relation, not explicitely defined in XBRL. The next section describe this ongoing work that we have recently started.

3 Translating XBRL Taxonomies into OWL

In this section, we report on our effort in translating XBRL taxonomies into an instance of description logic, viz., OWL, the Web Ontology Language [3]. We first introduce the basic tools and then move on to the translation process.

3.1 OWL, Protégé, and Sesame

XBRL taxonomies make use of XML Schema [4] in order to describe the structure of an XBRL document as well as to define new datatypes and properties, relevant to XBRL. Given such a schema and a validation program, it is then possible to check whether a concrete (business) document conforms to the syntactic structure, defined in the schema. As we have already indicated above, we probably need languages and tools that go beyond the expressive *syntactic* power of XML Schema.

OWL, the Web Ontology Language is the new emerging language for the Semantic Web that originates from the DAML+OIL standardisation. OWL still makes use of constructs from RDF [5] and RDFS [6], such as `rdf:resource`, `rdfs:subClassOf`, or `rdfs:domain`, but its two important variants OWL Lite and OWL DL restrict the expressive power of RDFS, thereby ensuring decidability. What makes OWL unique (as compared to RDFS or even XML Schema) is the fact that it can describe resources in more detail and that it comes with a well-defined model-theoretical semantics, inherited from description logic [7]. From description logic, OWL inherits further modelling constructs, such as `intersectionOf`, `equivalentClass`, or cardinality restrictions. The description logic background furthermore provides automated reasoning support such as consistency checking of the TBox and the ABox, subsumption checking during instance retrieval, etc.[3]

The XBRL OWL *base* taxonomy was manually developed using the OWL plugin of the Protégé knowledge base editor [8]. This version of Protégé comes with a partial OWL Lite support. The latest version of XBRL together with the Accounting Principles for German (our example, see below) consists of 2,414 concepts, 34 properties, and 4,780 instances. Overall, this translates into 24,395 unique RDF triples.

The basic idea during our effort was that even though we are developing an XBRL taxonomy in OWL using Protégé, the information that is stored on disk is still RDF on the syntactic level. We were thus interested in RDF data base systems which make sense of the semantics of OWL and RDFS constructs such as

[3] TBox and ABox are terms, introduced in the early days of description logic (or terminological logic; [?]). TBox refers to the terminological knowledge—knowledge about concepts that are relevant to our domain; for example, that `sharesItemType` is a subclass (or subconcept) of `shares`. In that sense, a TBox defines a domain schema. An ABox, however, represents assertions about individuals (of certain concepts), for instance, that `t_genInfo.doc.id` is related to `t_genInfo.doc` via the `partOf` property. Nowadays, the term ontology usually refers to both the TBox and the ABox.

`rdfs:subClassOf` or `owl:equivalentClass`. We currently experimenting with the Sesame open-source middleware framework for storing and retrieving RDF data [9]. Sesame partially supports the semantics of RDFS and OWL constructs via entailment rules that compute "missing" RDF triples (the deductive closure) in a forward-chaining style at compile time. Since sets of RDF statements represent RDF graphs, querying information in an RDF framework means to specify path expressions. Sesame comes with a very powerful query language, SeRQL, which includes (i) generalised path expressions, (ii) a restricted form of disjunction through optional matching, (iii) existential quantification over predicates, and (iv) Boolean constraints. From an RDF point of view, additional 62,598 triples were generated through Sesame's (incomplete) forward chaining inference mechanism. Let us give an example of a slightly simplified entailment rule that computes the missing triples with `hasPart` in predicate position (see also figure 1):

```
<rule name="owl-transitiveProp">
  <!-- note: ?p, ?x, ?y, and ?z are variables -->
  <premise>
    <subject var="?p"/>
    <predicate uri="&rdf;type"/>
    <object uri="&owl;TransitiveProperty"/>
  </premise>
  <premise>
    <subject var="?x"/>
    <predicate var="?p"/>
    <object var="?y"/>
  </premise>
  <premise>
    <subject var="?y"/>
    <predicate var="?p"/>
    <object var="?z"/>
  </premise>
  <consequent>
    <subject var="?x"/>
    <predicate var="?p"/>
    <object var="?z"/>
  </consequent>
</rule>
```

Since we have classified `hasPart` (as well as `partOf`) as a transitive OWL property, the above rule will fire, making implicit knowledge explicit and produces new triples such as

```
<t_bs, hasPart, t_bs.ass.defTax>
```

although only

```
<t_bs, hasPart, t_bs.ass>
<t_bs.ass, hasPart, t_bs.ass.defTax>
```

can be found in the original specification.

3.2 Translating the Base Taxonomy

For proof of concept, we looked at the freely available financial reporting taxonomies (`http://www.xbrl.org/FRTaxonomies/`) and took the final German *AP Commercial and Industrial* (German Accounting Principles) taxonomy (February 15, 2002; `http://www.xbrl-deutschland.de/xe_news2.htm`), acknowledged by XBRL International. The taxonomy can be optained as a packed zip file from `http://www.xbrl-deutschland.de/german_ap.zip`.

`xbrl-instance.xsd` specifies the XBRL base taxonomy using XML Schema. The file makes use of *XML schema datatypes*, such as `xsd:string` or `xsd:date`, but also defines *simple types* (`simpleType`), *complex types* (`complexType`), *elements* (`element`), and *attributes* (`attribute`). Element and attribute declarations are used to restrict the usage of elements and attributes in XBRL XML documents. Since OWL only knows the distinction between classes and properties, the correpondences between XBRL and OWL description primitives is not a one-to-one mapping:

XBRL	OWL
simple type	class
complex type	class
attribute	property
element	property

However, OWL allows to characterize properties more precisely than just having only a domain and a range. We can mark a property as *functional* (instead of being relational, the default case), meaning that it takes *at most* one value. This clearly means that a property must not have a value for each instance of a class on which it is defined. Thus a functional property is in fact a partial (and must not necessarily be a total) function. Exactly the distinction *functional vs. relational* is represented by the *attribute vs. element* distinction, since multiple elements are allowed within a surrounding context. However, at most one attribute-value combination for each attribute name is allowed within an element:

XBRL	OWL
simple type	class
complex type	class
attribute	functional property
element	relational property

Simple and complex types differs from one another in that simple types are essentially defined as extensions of the basic XML Schema datatypes, whereas complex types are XBRL specifications that do not build upon XSD types, but instead introduce their own element and attribute descriptions. Here are simple type specifications found in the base terminology of XBRL, located in file `xbrl-instance.xsd`:

```
    <attribute name="balance">
      <simpleType>
        <restriction base="string">
(1)       <enumeration value="debit"/>
          <enumeration value="credit"/>
        </restriction>
      </simpleType>
    </attribute>

    <simpleType name="monetary">
(2)   <restriction base="decimal"/>
    </simpleType>

    <simpleType name="dateUnion">
(3)   <union memberTypes="date dateTime"/>
    </simpleType>

    <simpleType name="operatorNameEnum">
      <restriction base="string">
(4)     <enumeration value="multiply"/>
        <enumeration value="divide"/>
      </restriction>
    </simpleType>
```

Let us have a closer look on the above five descriptions. First of all, let us compare (1) and (4). (4) specifies a type that is an extension of the base XSD type `string` in that it enumerate the only two possible extensions, viz., the strings `multiply` and `divide`. This definition is given a name, viz., `operatorNameEnum`. Contrary to this *named* "top-level" definion, (1) specifies a similar string enumeration type that is, however, *unnamed* (`name=` is missing). This unnamed definition is part of the *range* specification for attribute (OWL: property) `balance`. (2) defines a named extension of the XSD base type `decimal`. Finally, (3) defines `dateUnion` to be the union of the two XSD types `date` and `dateTime`.

At this point, let us have a few remarks on the representation of the XSD base types in our implementation. Since OWL only claims that

> *As a minimum, tools* [using OWL] *must support datatype reasoning for the XML Schema datatypes* `xsd:string` *and* `xsd:integer`*.* [3, p. 30]

and because

> *It is not illegal, although* **not** *recommended, for applications to define their own datatypes ...* [3, p. 29]

we have decided to implement a workaround that represents all the necessary XML Schema datatypes used in XBRL. This was done by having a wrapper type for each simple XML Schema type. For instance, `monetray` (2) is a simple subtype of the wrapper type `decimal`: `<restriction base="decimal"/>`.

Now comes the definition of `dateUnion` (3). OWL DL's `unionOf` construct exactly models the intended meaning: $\texttt{dateUnion} \equiv \texttt{date} \sqcup \texttt{dateTime}$. Since `dateUnion` is also a subtype of `Xbrli` (the most general complex type), we further have $\texttt{dateUnion} \sqsubseteq \texttt{Xbrli}$. Since `dateUnion` is exactly the union of `date` and `dateTime`, OWL will further infer that $\texttt{dateUnion} \sqsubseteq \texttt{anySimpleType}$, since we know that $\texttt{date} \sqsubseteq \texttt{anySimpleType}$ and $\texttt{dateTime} \sqsubseteq \texttt{anySimpleType}$ is the case.

Finally, let us report on the final definition (4) of `operatorNameEnum`. OWL DL provides `oneOf` that supports user-defined enumerated datatypes, types which are solely defined through their finite number of instances (contrary to the potential infinite datatypes, e.g., `decimal`). Thus we have $\texttt{operatorNameEnum} \equiv \{\texttt{multiply}, \texttt{divide}\}$, but also at the same time $\texttt{operatorNameEnum} \sqsubseteq \texttt{string}$ and $\texttt{operatorNameEnum} \sqsubseteq \texttt{Xbrli}$.

Example (1) is somewhat different from (2)–(4) in that it uses an *unnamed* type that is further used inside an attribute description. This property `balance` is silent about its domain and, in principle, might be applied to each instance, so we can assume `owl:Thing` here. But `balance` is very concrete about its range: the enumeration of `debit` and `credit`: $\texttt{balance} \sqsubseteq \langle \texttt{owl} : \texttt{Thing}, \{\texttt{debit}, \texttt{credit}\}\rangle$, together with the restriction that `balance` is a functional property (since it is defined as an XML Schema attribute): $\texttt{balance} = 0 \sqcup \texttt{balance} = 1$. The informal OWL description used so far makes a loose reference to the Protégé OWL syntax.

The translation of complex types is more intricate. Take, for instance, the specification of `monetaryItemType` that extends `monetary`:

```
     <complexType name="monetaryItemType">
       <simpleContent>
         <extension base="xbrli:monetary">
(5)        <attribute name="numericContext" type="IDREF" use="required"/>
         </extension>
       </simpleContent>
     </complexType>
```

This type, as well as `sharesItemType` and `decimalItemType`, refer through its attribute `numericContext` via IDREF to an element `numericContext`:

```
     <element name="numericContext">
       <complexType>
         <sequence>
           <element name="entity" type="xbrli:entityType"/>
           <element name="period" type="xbrli:periodType"/>
           <element name="unit" type="xbrli:unitType"/>
(6)        <element ref="xbrli:scenario" minOccurs="0"/>
         </sequence>
         <attribute name="id" type="ID" use="required"/>
         <attribute name="precision" type="string" use="required"/>
         <attribute name="cwa" type="boolean" use="required"/>
       </complexType>
     </element>
```

Since `numericContext` is solely defined on the above three types, its domain becomes the union of those types:

$$\texttt{monetaryItemType} \sqcup \texttt{sharesItemType} \sqcup \texttt{decimalItemType}$$

The use of `IDREF` in the above three complex type definitions is simply restated by saying that `numericContext` is defined on those types. The additional constraint `use="required"` is represented by adding a *local* class restriction, saying that the cardinality of `numericContext` is 1 for the three types `monetaryItemType`, `sharesItemType`, and `decimalItemType`. Recall, that since `numericContext` is used as an attribute in the three item types, a cardinality of 0 is in principle possible. Note, that this cardinality restriction does not hold for `numericContext` in general—it is still an element and thus modelled in OWL as a relational property. Again, we have an element `numericContext` that has a top-level definition, saying that it maps onto objects of an unnamed type, consisting of properties, such as `entity` or `cwa`. Since OWL does not directly comes up with a `sequence`[4] construct, we have simplified the definition of `numericContext` in that we put the above four sequence elements on par with the other three. This is reasonable since elements have a name and no name conflict is possible at this point.

It should now be clear how the range of `numericContext` looks like. Given Protégé's OWL syntax and using OWL's `allValuesFrom` (∀), `someValuesFrom` (∃), and cardinality restrictions, we have

$$\begin{array}{l}
(\forall\ \texttt{entity entityType}) \sqcap \\
(\forall\ \texttt{period periodType}) \sqcap \\
(\forall\ \texttt{unit unitType}) \sqcap \\
(\texttt{scenario} \geq 0) \sqcap \\
(\exists\ \texttt{precision string}) \sqcap \\
(\texttt{precision} = 1) \sqcap \\
(\exists\ \texttt{cwa boolean}) \sqcap \\
(\texttt{boolean} = 1)
\end{array}$$

Note that the attribute `id` of type `ID` is not remodelled here, since referencing `numericContext` in the above three complex types (`IDREF`!) is done by simply using the name of this attribute.

Let us finally come to another "problematic" XML Schema construct that does not has a direct counterpart in OWL: `choice`, a kind of XOR. Consider, for instance, the element definition of `operator`:

```
<element name="operator">
  <complexType>
    <choice minOccurs="2" maxOccurs="2">
      <element ref="measure"/>
```

[4] Of course, we could have used a `FIRST-REST` list encoding, but such a construction would make the definition more complex than is really needed.

```
(7)        <element ref="operator"/>
         </choice>
         <attribute name="name" type="operatorNameEnum" use="required"/>
       </complexType>
     </element>
```

This recursive operator expression defines a special kind of balanced binary trees and can clearly be modelled using AND ($\sqcap$), OR ($\sqcup$), and cardinality restrictions (or negation, in the general case), and in fact, we have defined the range of `operator` as follows (`name` itself is already defined as a functional property):

$$(\exists\,\texttt{name}\ \texttt{operatorNameEnum}) \sqcap$$
$$(((\texttt{measure} = 2) \sqcap (\texttt{operator} = 0)) \sqcup ((\texttt{measure} = 0) \sqcap (\texttt{operator} = 2)))$$

However, there exist examples with more than two elements involved in a choice:

```
     <complexType name="periodType">
       <choice>
         <sequence minOccurs="0">
           <element ref="xbrli:startDate"/>
           <choice>
             <element ref="xbrli:endDate"/>
             <element ref="xbrli:duration"/>
           </choice>
(8)      </sequence>
         <sequence minOccurs="0">
           <element ref="xbrli:duration"/>
           <element ref="xbrli:endDate"/>
         </sequence>
         <element ref="xbrli:instant" minOccurs="0"/>
         <element ref="xbrli:forever" minOccurs="0"/>
       </choice>
     </complexType>
```

A proper approximation in OWL avoiding the `sequence` construct would be the following unhandy class expression:

$$((\texttt{startDate} = 1) \sqcap (\texttt{endDate} = 1) \sqcap (\texttt{duration} = 0) \sqcap (\texttt{instant} = 0) \sqcap (\texttt{forever} = 0)) \sqcup$$
$$((\texttt{startDate} = 1) \sqcap (\texttt{duration} = 1) \sqcap (\texttt{endDate} = 0) \sqcap (\texttt{instant} = 0) \sqcap (\texttt{forever} = 0)) \sqcup$$
$$((\texttt{duration} = 1) \sqcap (\texttt{endDate} = 1) \sqcap (\texttt{startDate} = 0) \sqcap (\texttt{instant} = 0) \sqcap (\texttt{forever} = 0)) \sqcup$$
$$((\texttt{instant} = 1) \sqcap (\texttt{startDate} = 0) \sqcap (\texttt{endDate} = 0) \sqcap (\texttt{duration} = 0) \sqcap (\texttt{forever} = 0)) \sqcup$$
$$((\texttt{forever} = 1) \sqcap (\texttt{startDate} = 0) \sqcap (\texttt{endDate} = 0) \sqcap (\texttt{duration} = 0) \sqcap (\texttt{instant} = 0))$$

At the moment, we have opted to modell such a definition by simply listing the already defined properties (6)–(10) for the OWL concept `periodType`.

3.3 Translating the Accounting Principles

The last section has explained how the base taxonomy of XBRL is represented. This section is devoted to the translation of the German Accounting Principles (AP; see `http://www.xbrl-deutschland.de/xe_news2.htm`). At the moment, we have taken the following three files into account:

1. `german_ap_definition.xml`
2. `german_ap_label.xml`
3. `german_ap.xsd`

The file `german_ap_definition.xml` basically defines a *part-of*/*has-part* taxonomy via the XML Linking Language XLink [10] through the use of `xlink:from` and `xlink:to` in `definitionArc` elements.[5] Note that we have characterized the `partOf` and `hasPart` properties in OWL as mutual *inverse* and *transitive*, important for later reasoning.

`german_ap_definition.xml` also specifies the basic objects of interest, the so-called *locators* (`xlink:type="locator"`) via `loc` elements. Such a `loc` element is furthermore equipped with three attributes:

1. `xlink:href`: the URI reference for this class/concept
2. `xlink:title`: the name of the class/concept
3. `xlink:label`: the name of the instance of the class/concept

For each AP concept, we define a unique instance that can be used in the description of other AP concept instances. This way of representing information is due to the fact that OWL (or description logic in general) enforces properties, such as `partOf`, to connect individuals, but not classes (we are talking about the ABox here!).

`german_ap_label.xml` provides further information concerning a natural language description of the concept, both in German (`xml:lang="de"`) and English (`xml:lang="en"`), using `label` elements. This file also respecifies the part-of/has-part taxonomy.

`german_ap.xsd` finally links the AP concepts to concepts defined in the base taxonomy, saying that, for instance, `bs.ass.fixAss` is of type `monetary`: `type="xbrli:monetary"`; see definition (2) for `monetary`.

Overall, a locator object thus consists of the following six properties:

1. `deLabel` (single `xsd:string`)
2. `enLabel` (single `xsd:string`)
3. `href` (single `xsd:string`)
4. `type` (single `Xbrli`)
5. `partOf` (multiple `Locator`)
6. `hasPart` (multiple `Locator`)

The outcome of our translation of the (single) instance `t_bs.ass` for the XBRLI concept `bs.ass` (total assets) is depicted in the next figure.

[5] At first sight, one might think that `german_ap_definition.xml` defines a sub-/supertype taxonomy.

t_bs.ass	
deLabel	Summe Aktiva
enLabel	Total Assets
href	german_ap.xsd#bs.ass
type	xbrli_monetaryItemType_230
partOf	t_bs
hasPart	t_bs.ass.defTax
	t_bs.ass.assInbetwFixAndCurr
	t_bs.ass.deficitNotCoveredByCapital
	t_bs.ass.accountingConvenience
	t_bs.ass.other
	t_bs.ass.prepaidExp
	t_bs.ass.currAss
	t_bs.ass.fixAss
	t_bs.ass.unpaidCap

Fig. 1. The internal structure of the instance `t_bs.ass`, representing XBRL's insight on the notion of *total assets*. `t_bs.ass` refers to other instances via its `hasPart`/`partOf` properties. `type`, `partOf`, and `hasPart` are OWL object properties, whereas `deLabel`, `enLabel`, and `href` are so-called datatype properties.

Overall, the German Accounting Principles taxonomy consists of 2,387 concepts, plus 27 concepts from the base taxonomy for XBRL. 34 properties were defined and 4,780 instance finally generated. The running time of the translation process is about 3 seconds on a mid-size Linux machine under Java 2 SE 5.0.

4 Summary

This paper has presented a translation schema for the base ontology of XBRL into OWL that was extended to the German Accounting principles. Our enterprise was initiated by work in the eTen WINS project that needs reasoning support going beyond the syntactic power of XML Schema. We believe that our work is a first step towards large-scale intelligent web-based financial information and decision support systems.

References

1. Fornasari, F., Tommasi, M.N.A., Zavattari, C., Gagliardi, R., Declerck, T.: XBRL web-based business intelligence services. In: Proceedings of the rChallenge Conference 05, Springer (2005)
2. Hobbs, J., Pan, F.: An ontology of time for the Semantic Web. ACM Transactions on Asian Language Processing (TALIP) **3**(1) (2004) 66–85
3. Bechhofer, S., van Harmelen, F., Hendler, J., Horrocks, I., McGuinness, D.L., Patel-Schneider, P.F., Stein, L.A.: OWL web ontology language reference. Technical report, W3C (2004) 10 February.

4. Fallside, D.C., Walmsley, P.: XML schema part 0: Primer second edition. Technical report, W3C (2004) 28 October.
5. Klyne, G., Carroll, J.J.: Resource description framework (RDF): Concepts and abstract syntax. Technical report, W3C (2004) 10 February.
6. Brickley, D., Guha, R.V.: RDF vocabulary description language 1.0: RDF Schema. Technical report, W3C (2004) 10 February.
7. Baader, F., Calvanese, D., McGuinness, D., Nardi, D., Patel-Schneider, P.: The Description Logic Handbook. Cambridge University Press, Cambridge (2003)
8. Knublauch, H., Musen, M.A., Rector, A.L.: Editing description logic ontologies with the Protégé OWL plugin. In: Proceedings of the International Workshop on Description Logics, DL2004. (2004)
9. Broekstra, J., Kampman, A., van Harmelen, F.: Sesame: A generic archistecture for storing and querying RDF and RDF schema. In: Proceedings of the International Semantic Web Conference (ISWC). Number 2342 in Lecture Notes in Computer Science (LNCS), Springer (2002) 54–68
10. DeRose, S., Maler, E., Orchard, D.: XML Linking Language (XLink) Version 1.0. Technical report, W3C (2001) 27 June.