Martin Hepp
A Methodology for Deriving OWL Ontologies from Products and Services Categorization Standards
Proceedings of the 13th European Conference on Information Systems (ECIS2005), May 26 - May 28, 2005, Regensburg, Germany, pp. 1-12.

A METHODOLOGY FOR DERIVING OWL ONTOLOGIES FROM PRODUCTS AND SERVICES CATEGORIZATION STANDARDS

Hepp, Martin, Florida Gulf Coast University, Department of CIS, College of Business, 10501 FGCU Blvd South, Fort Myers, FL 33965-6565, USA; Digital Enterprise Research Institute, Leopold-Franzens University of Innsbruck, Technikerstraße 21a, AT-6020 Innsbruck, Austria, mhepp@computer.org

Abstract

Products and services categorization standards (PSCS), like UNSPSC, eCl@ss, eOTD, or the RosettaNet Technical Dictionary (RNTD) form a valuable set of concepts from the product and services domain and reflect some degree of consensus. They are thus a promising foundation for the creation of products and services ontologies. Existing approaches for this task, however, do neither properly reflect the specific semantics of the respective categorization standards, nor do they sufficiently address the high versioning dynamics due to product innovation. This paper presents a comprehensive approach for the proper reuse of such standards in product ontologies

Keywords: Products and Services Ontologies, Taxonomies, OWL, UNSPSC, eCl@ss

1 INTRODUCTION

Products and services categorization standards (PSCS), like the UNSPSC, eCl@ss, eOTD, or the RosettaNet Technical Dictionary (RNTD) form a valuable set of concepts from the products and services domain and reflect some degree of consensus. They are thus a promising foundation for the creation of product ontologies for the use in the Semantic Web, but are not by themselves fully usable ontologies.

There exist already examples of products and services ontologies derived from PSCS, e.g. the DAML version by McGuinness (2001) and the RDF-S vocabulary version of UNSPSC by Klein (2002). However, those ontologies do not properly reflect the specific semantics of the underlying standards, nor do they provide support for dealing with versioning dynamics. Both shortcomings in combination limit the usefulness of those ontologies to a very narrow set of applications.

Ontology engineering has been the subject of research for a long time and multiple methodologies for the creation of ontologies have been proposed. However, the script-based (i.e. automated) reuse of concepts from hierarchically ordered standards, which were not created with the rigor of knowledge representation in mind, creates different requirements and demands novel approaches, mainly because we cannot partition the concept space into concepts at our own will. In other words, we must automatically capture as much semantics as possible out of the existing standards and have only limited control over the actual definition of the concepts. Basically, we have to take the concepts as they are, because the size of those standards (between 20,000 and almost 60,000 classes and between 3,000 and 20,000 properties) prohibits manual steps in the process of importing the concepts. On the other hand, we have to carefully analyze the constraints and dependencies resulting from the implicit assumptions of the standards creators. Additionally, the intension of the exported concepts will often be influenced by our interpretation of the original standard. If we cannot fully avoid changes in meaning between the original standard and the resulting ontology, then we must at least make respective decisions transparent to the ontology user.

1.1 Related Work

Related work can be classified into the following main groups:

- Ontology engineering methodologies, implicitly or explicitly focusing on the manual creation of ontologies based on knowledge engineering principles. A comprehensive discussion of all approaches in this field is beyond the scope of this paper, for an overview see e.g. Fernández-López and Gómez-Pérez (2002) and de Bruijn (2003).

- Analysis of the meaning of taxonomic relationships, especially the fundamental work of Brachman (1983). This yielded the insight that there are multiple types of taxonomic relationships, which should be represented separately.
- Contributions regarding Electronic Commerce ontologies mainly from the perspective of catalog data management and ontology mapping. Fensel et al. (2001b) discuss the advantages of ontologies for the integration of heterogeneous and distributed information in the Electronic Commerce field. Schulten et al. (2001) and Fensel et al. (2001a) focus on the problem of product data integration in B2B relationships. Ng et al. (2000) introduce the important distinction between content standardization versus content integration as two approaches for overcoming information heterogeneity. Fairchild and de Vuyst (2002) detail the benefits of standardized products and services coding from a business perspective. Obrst et al. (2001) discuss the challenges associated with the creation of ontologies for the product and service knowledge space, stressing the task of mapping among ontologies. Kim et al. (2002) propose catalog data integration based on the controlled merging of category hierarchies. Kim et al. (2004) discuss the problematic aspects of a naïve view on product classification as one "ideal" way of ordering and introduces the important notion of any classification being a purpose-specific order and thus not universally useable.
- Prototypes of products and services ontologies in standard ontology languages derived from UNSPSC. To our knowledge, there are currently two examples of UNSPSC transformations into ontology representation languages: The DAML+OIL and RDF-S variants created by Klein (2002) and the DAML+OIL variant from the Knowledge Systems Laboratory at Stanford.
- Methodologies for and experiences with the reuse of consensus in existing standards for the creation of ontologies. This is the most relevant field of work for this paper. Rector et al. (2001) discuss the transformation of tangled hierarchies, as e.g. such derived from ambiguous "broader than / narrower than" taxonomies in library science, into formal ontologies. Kashyap et al. (2003) describe the experiences gained while transforming the constructs of an existing semantic network in the medical domain into an OWL ontology. Wielinga et al. (2001) show the reuse and semantic enrichment of an existing taxonomy and demonstrates this for the Art and Architecture Thesaurus (AAT). Wielinga et al. (2004) and van Assem et al. (2004) are consequent works of this stream of research. An important characteristic of Wielinga et al. (2004) and van Assem et al. (2004) is that they leave the limits of OWL DL in order to capture semantics contained in the original taxonomy, namely to be able to treat classes as instances and vice versa. An important distinction between the proposal of Wielinga et al. (2004) and the work presented in our paper is that they treat the taxonomic relationship in the input taxonomy as a special form (i.e. rdfs:subPropertyOf) of rdfs:subClassOf, whereas in our approach, the original taxonomic relationship is regarded as a weaker, more general, and semantically less specific relationship. In other words, we regard rdfs:subClassOf as a specific form of a general taxonomic relationship, i.e. a rdfs:subPropertyOf of the more general idea of a taxonomic relationship. Section 3.1 explains this approach in greater detail.

1.2 Our Contribution

In this paper, we first show that existing DAML and RDF-S domain ontologies derived from products and services standards, namely UNSPSC, are based on incorrect assumptions about the meaning of the taxonomic relationship in those standards, rendering the resulting ontologies almost useless. Second, we present a new methodology that allows to properly reflect the taxonomic relationship of PSCS in an OWL Lite ontology, yielding a fully-fledged products and services ontology. Third, we develop a consistent versioning mechanism that is compatible with existing OWL Lite constructs. Fourth, we show how the valuable property recommendations for classes and the value recommendations for properties can be represented without violating the limits of OWL Lite and the Open World Assumption (OWA). Fifth, we demonstrate the application of our novel approach to eCl@ss 5.0 and provide a comprehensive ontology for the products and services domain. To our knowledge, the resulting ontology is the first real-world domain ontology for products and services.

The structure of the paper is as follows. Section 2 introduces the components and architecture of prominent products and services categorization standards (PSCS). Section 3 presents a proposal on how existing PSCS can be transformed into OWL ontologies, preserving the specific semantics of those standards. Section 4 illustrates the proposal by applying it to the standard eCl@ss. Section 5 evaluates this novel approach, compares it to previous work, summarizes the results, and presents the major conclusions.

2 COMPONENTS OF PRODUCTS AND SERVICES CATEGORIZATION STANDARDS

There are countless approaches for the classification of goods, ranging from rather coarse taxonomies, created for customs purposes and statistics of economic activities, like the North American Industry Classification System (NAICS) and its predecessor SIC (see U.S. Census Bureau (2004)), to expressive descriptive languages for products and services, like eCl@ss, eOTD, or the RosettaNet Technical Dictionary. The UNSPSC, widely cited as an example of a product ontology, is in the middle between those two extremes, providing an industry-neutral taxonomy of products and services categories, but no standardized properties for the detailed description of products.

It is out of the scope of this paper to list and compare all available standards in this area, but one can say that UNSPSC, eCl@ss, and eOTD are currently the most important horizontal standards (i.e. covering a broad range of industries), and RNTD should be included in the analysis because of its high degree of detail, albeit limited to a narrow segment of products.

All of those standards reflect a varying combination of the following components, which can be reused for a derived ontology. It is crucial to observe that most standards were not created with the rigor of ontology engineering, or knowledge representation in general, in mind, but for very specific purposes, leading to several implicit assumptions. Failure to understand the context of the underlying standard and its intended usage yields inconsistent and almost useless ontologies.

Product Classes: All PSCS are centered around a set of product categories that try to group similar products. However, this grouping is often influenced by the purpose of the PSCS. For example, the categories can try to collect products by the *nature* of the products or by their intended *usage*. This is most obvious when it comes to classifying chemicals: The same substance can be used for multiple purposes, and classification systems can be based on classes for the chemical substances, categories for the various usages, or a combination of both. This can create confusion, as there is an N:M mapping between the nature of a product and product usages.

The intensions of the product classes are usually captured in a rather informal way, ranging from just very short class names to quite precise natural language definitions, sometimes available in multiple languages.

Hierarchy of Classes: Most PSCS arrange the classes in hierarchical order. It is crucial to understand that this hierarchy is directly connected to the intended usage of the PSCS. For example, eCl@ss was designed with the idea of grouping products from the perspective of a buying organization or a purchasing manager. So it regards products as similar that are (1) bought from the same range of suppliers, (2) needed by the same consumers inside the organization, (3) billed using the same cost accounting categories, or any combination of those. A typical consequence of this is that related classes (e.g. service or maintenance) are often subclasses of the general good category. The category “TV Set Maintenance” will thus be regularly a subclass of “TV Sets”, and “Oil for Sewing Machines” will be a subclass of “Sewing Machinery”. As a summary, the semantics of this relationship is generally not more specific than “Class A is some special kind of Class B”.

Dictionary of Properties: More sophisticated PSCS include a dictionary of standardized properties that can be used to describe product instances or product models in more detail and allow parametric search. Usually, those property dictionaries contain a quite rich definition of the contained properties, including not only sophisticated data typing, but also references to international standards for the unit of measurement. From the perspective of ontology engineering, most of the properties are data type properties, though there are also recommended sets of enumerated values for a small part of the properties.

Enumerated Property Values: For properties where an arbitrary string is not sufficient to capture the value in a semantically unambiguous way, most PSCS maintain a list of supported values in a separate collection. For example, eCl@ss contains the property “Type of door” (ID AAA434001 in version 5.0SP1), with the lexical space set to an arbitrary string of up to 15 characters. The set of supported values contains two entries that are recommended for the use with this property, “Folding doors” (ID AAA376001) and “Sliding doors” (ID AAA377001). The mapping between recommended values and such properties is usually kept in a separate relation.

Class-Property Relation: Most PSCS with a dictionary of properties include a mapping between classes and recommended properties, i.e. property sets per each class, sometimes referred to as “attribute lists”. However, the semantics of this assignment varies between different standards. It can range from very loose recommendations (as in eOTD) to a strict definition of those properties necessary and sufficient to completely describe an instance of the respective class (as in eCl@ss). The creation and maintenance of such property assignments is a tremendous task, and none of the existing horizontal PSCS has specific property lists for more than 50 % of their categories (see Hepp et al. (2004) for a comprehensive analysis).

Due to the continuous innovation in the product and services domain, all PSCS are a work in progress with multiple releases per year. In a recent analysis we found out that, on average, eCl@ss grows by as much as 280 and UNSPSC by 230 new classes per 30 days. As a consequence, transforming a PSCS into an ontology is no one time task, but a recurring activity.

3 DERIVING OWL ONTOLOGIES FROM PSCS

The basic challenge when deriving ontologies from PSCS is to represent as much as possible of the original semantics in the taxonomy while using the allowed constructs of the respective ontology language. In this section we explain how previous attempts fail to properly reflect the semantics of the reused PSCS, especially with regard to the taxonomic relationship, propose a novel methodology, and show in detail how the core components of PSCS (as introduced in section 2) can be properly represented in OWL Lite.

3.1 Product Classes and Hierarchy

When taking the categories found in a taxonomy as the basis for the creation of an ontology, we face a fundamental -- but so far ignored -- problem: Unless there is a formal definition of the semantics of the taxonomic relationship, the intensions of the category concepts (e.g. the product classes) *are not determined independently from the interpretation of the taxonomic relationship.* In other words: If we lack a formal definition of either the hierarchical relationships or the category concepts, then *how we understand the taxonomic relationship determines the shape of the category concepts* and vice versa. Our choice of the interpretation of the taxonomic relationship affects the intension of the category concepts, and a chosen definition of the intension of the category concepts is compatible with only a specific interpretation of the taxonomic relationship.

As a consequence, we have some degree of choice over the intension of the ontology classes derived from the categories in the source taxonomy by selecting the interpretation of the taxonomic relationship.

Two examples might illustrate this fundamental problem: The hierarchies of both UNSPSC and eCl@ss were created on the basis of practical aspects of procurement, treating those commodities that “somehow” belong to a specific category as descendents of this closest category. This makes “ice” a subclass of “non-alcoholic beverages” in UNSPSC and “docking stations” a subcategory of “computers” in eCl@ss. Now, we still can read the taxonomic relationship as a strict “rdfs:subClassOf” relationship (i.e. each instance of “ice” is also an instance of “non-alcoholic beverages” and each instance of “docking station” is also an instance of “computers”). Then, however, the intension of the class “computers” is no longer any computer, but the concept “computer” solely from the perspective of cost accounting or spend analysis, where an incoming invoice for a docking station can be treated as an incoming invoice for a computer. Similarly will "non-alcoholic beverages" no longer represent all non-alcoholic beverages, but the union of non-alcoholic beverages and related commodities.

The fatal consequence of this interpretation of the taxonomic relationship as being equivalent to rdfs:subClassOf is obvious: We can no longer use the resulting classes for buying processes, because a search for all instances of "computers" will also return docking stations, and ordering the cheapest available instance of non-alcoholic beverages will very likely return just ice cubes.

Given now the fact that our interpretation of the taxonomic relationship in the source taxonomy determines the intension of the classes, we have to *deliberately select* the most useful shape of the product classes and then derive the required interpretation for the taxonomic relationship. Basically, each source taxonomy contains two concepts for each category node: First the generic concept of the respective product or service category (e.g. "computer", marked as (1) in Figure 1) and second the intersection (marked as (2) in Figure 1) of this concept with a concept "element in this taxonomy" (marked as (3) in Figure 1), the later reflecting all the implicit assumptions of the creators of the taxonomy and the constraints resulting from the interpretation of the taxonomic relationship.

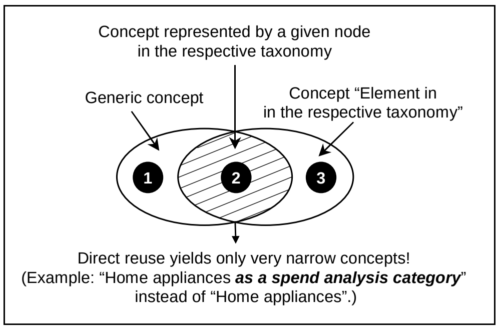
Concept represented by a given node in the respective taxonomy
Generic concept
Concept "Element in in the respective taxonomy"
1
2
3
Direct reuse yields only very narrow concepts!
(Example: "Home appliances ***as a spend analysis category***" instead of "Home appliances".)

Figure 1: Relationship between generic concepts and taxonomy nodes

Now, even though the taxonomic relationship should not be interpreted as rdfs:subClassOf, we still want to capture the relationship as such, because there are applications like cost accounting where it is very useful. So we have to find a way to represent both generic categories for general use, while still preserving the hierarchical order for analytical purposes and similar applications.

Basically there are at least the following four approaches of transforming a given PSCS into an OWL Lite ontology, of which only the last three meet the stated requirements.

1. Create one class for each taxonomy category and assume that the meaning of the taxonomic relationship is equivalent to rdfs:subClassOf.
2. Create one class for each taxonomy category and represent the taxonomic relationship using an *annotation* property taxonomySubClassOf.
3. Treat the category concepts as instances instead of classes and connect them using a transitive object property taxonomySubClassOf.
4. Create two concepts for each taxonomy category, one reflecting the generic product or service type and another reflecting the taxonomy concept.

Approach 1 is chosen by both available transformations of UNSPSC into products and services ontologies. For example, the RDF Schema representation of UNSPSC created by Klein (2002) contains statements of the following kind:

```
<rdfs:Class rdf:ID="Ice">
  <rdf:type rdf:resource="http://ontoview.org/schema/unspsc/1#Commodity"/>
  <rdfs:subClassOf rdf:resource="#Non_alcoholic_beverages"/>
  <unspsc:egci>014067</unspsc:egci>
  <unspsc:code>50.20.23.02</unspsc:code>
</rdfs:Class>
```

The DAML representation of UNSPSC created by the Knowledge Systems Laboratory at Stanford University McGuinness (2001) uses the same structure.

While the underlying approach is not necessarily incorrect, it does not yield a products and services ontology, but a set of cost accounting and purchasing management categories. Quite clearly, we want to make the resulting products and services ontology be useful for many different application areas, including the search for products and services, and not limit the usage to spend analysis.

Solution 2 seems to be the most straightforward alternative, since the specific meaning of the taxonomic relationships is captured using a specific property and the classes can represent generic product concepts. The problem with this approach is that, in OWL Lite and OWL DL, a property that connects classes with classes can only be an annotation property. Thus, it cannot be made a transitive property, and a OWL Lite or OWL DL reasoner will only see explicit statements. In other words, if class A is a taxonomySubClassOf class B and class B is a taxonomySubClassOf C, the reasoner will not infer that class A is also a taxonomySubClassOf class C.

This limitation can be avoided by making the products and services concepts instances instead of classes, as described in solution 3. Then, the property “taxonomySubClassOf” can be an owl:ObjectProperty and can be made transitive. The downside of this approach is that one absolutely needs OWL Lite or OWL DL reasoning support in order to process the transitive nature of the property. For reasons detailed below, we wanted to find a solution that works with the “intersection” of RDF-S and OWL Lite, i.e. that does not require reasoning capabilities beyond rdfs:subClassOf.

It might seem strange from a pure modeling perspective to impose both the limitations of OWL Lite (e.g. classes cannot be instances) and the limited reasoning of RDF-S (no transitive properties other than rdfs:subClassOf and rdfs:subPropertyOf). However, this makes sense from a practical perspective, for the following reasons:

- It works with faster but incomplete implementations of OWL reasoners, especially the use of an RDF-S reasoner on OWL ontologies (supported by some repositories), while being upward compatible to future implementations. Especially, it allows for a merge with other OWL Lite data without making the result of the merge leave the boundaries of OWL Lite.
- It is possible to create an RDF-S variant of the ontology using the same constructs with only minimal changes. The low performance of current OWL implementations might make this step back necessary. We have to keep in mind that the resulting ontologies are rather big (tens of megabytes).

This leaves solution 4, which we currently favor. The basic idea is as following:

1. We create two separate concepts for (1) the *generic* product or service category and (2) the respective *taxonomy* category.
2. We arrange the *taxonomy* concepts in a strict rdfs:subClassOf hierarchy, but not the generic concepts. This allows for capturing the hierarchy of taxonomy concepts without linking the generic concepts to incorrect superordinate classes.
3. In order to ease annotation, we create one annotation class for each taxonomy node which becomes an rdfs:subClassOf *both* the respective generic and the respective taxonomy concept. With this construct, a single rdf:type statement is sufficient to make a product an instance of both the generic and the taxonomy concept.

Figure 2 illustrates this approach. The concept numbering with (1) and (2) refers to the numbering in Figure 1. The application of this approach for describing products is shown in Figure 3 (left side): The TV maintenance service ex:tv-set-repair is an instance of the annotation class “TV Set Maintenance”. This makes it also an instance of the generic product class “TV Set Maintenance (Generic)” and the taxonomy concept “TV Set Maintenance (Taxonomy)”. The second is a subclass of “TV Set (Taxonomy)”, but the first is not a subclass of “TV Set (Generic)”. This yields exactly the distinction we want: When searching for a TV maintenance service, we look for instances of the *generic* class, and when looking for all items that belong to the taxonomy category, we use the taxonomy concept. For example, a store manager might want to find all products in the TV set segment. In this case, he or she also wants to find TV set cabling and maintenance, so the query will be based on the taxonomy concept.

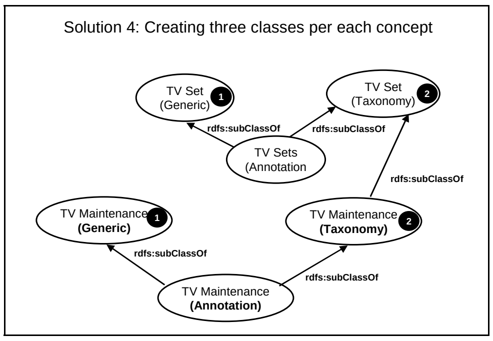
Solution 4: Creating three classes per each concept
TV Set (Generic) 1
TV Set (Taxonomy) 2
TV Sets (Annotation
rdfs:subClassOf
TV Maintenance (Generic) 1
TV Maintenance (Taxonomy) 2
TV Maintenance (Annotation)

Figure 2: Alternate solution: Separating the generic concept from the taxonomic concept

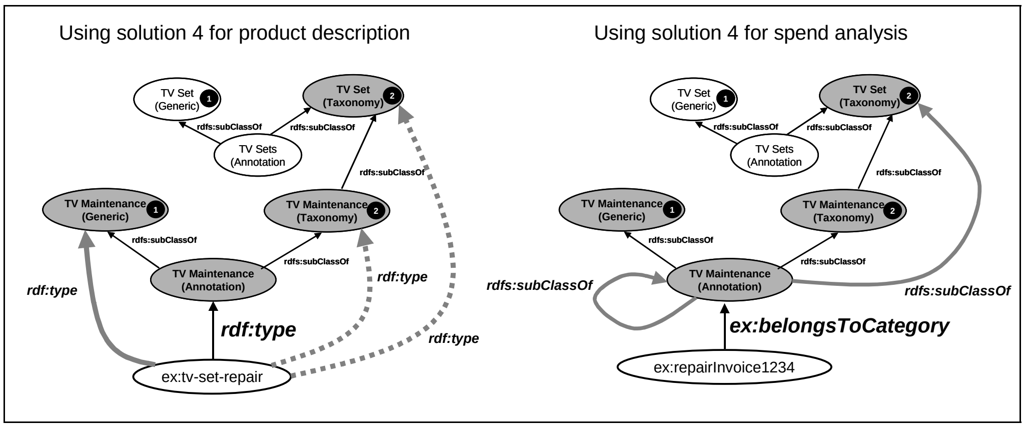
Using solution 4 for product description
Using solution 4 for spend analysis
TV Set (Generic) 1
TV Set (Taxonomy) 2
TV Sets (Annotation
rdfs:subClassOf
TV Maintenance (Generic) 1
TV Maintenance (Taxonomy) 2
TV Maintenance (Annotation)
rdf:type
ex:tv-set-repair
ex:belongsToCategory
ex:repairInvoice1234

Figure 3: Usage for product description and for tagging incoming invoices for spend analysis

The resulting ontology is not limited to describing product instances, but can also be used for annotating any kind of other entities. For example, incoming invoices can be linked to the respective classes with an annotation property “ex:belongsToCategory” (Figure 3, right). This allows to capture the fact that a specific invoice is referring to the respective category, but is not an instance. As every class in OWL is also an rdfs:subClassOf itself and rdfs:subClassOf is transitive, one can easily search for all resources belonging to a taxonomy category on any level of the hierarchy (see section 4.2 for the example of an RDQL query).

3.2 Product Properties and Property Data Typing

From an ontology engineering perspective, the set of properties in PSCS might seem rather trivial, because most of them are simple data type properties. However, their contribution to machine readable semantics in the Semantic Web is huge, for they provide standardized representation for concepts as generic as “weight” or as specific as “pump capacity”. Some of them can also be applied usefully when the class of a product is not known or no proper class exists. For example, it might be helpful to represent the weight of a novel product even though we do not yet have a proper class for this product.

The import of properties from PSCS into an OWL Lite ontology requires the following steps:

1. Determine whether it is a data type property or an object property, i.e. whether there are enumerated data values for this property or not.
2. Map the data type of the property to one XSD data type supported by current reasoners.

3. Add the unit of measurement (e.g. "inches"), usually stored separately in the PSCS, to the description of the property.
4. Create respective OWL properties for each property contained in the source PSCS.
5. Create instances of a new class "PropertyValue" for each enumerated data value.

It is important to know that the data types used in the PSCS cannot be directly mapped to standard XSD data types. eCl@ss, for example, uses over hundred different data type definitions based on ISO 9735 and ISO 6093, which are far more specific than available standard XSD data types. There are basically two alternatives for this transformation:

- One can use a rather coarse mapping from the specific ISO 9735 and ISO 6093 data types to standard XSD data types, loosing constraints of the original definition (e.g. field length, number of digits, …).
- Alternatively, it seems possible to create specific XSD data types or use constraining facets (see W3C (2001), section 4.3) to further constrain the standard XSD data types.

Given the yet limited support of reasoning for data types in Semantic Web tools, the first approach seems currently more appropriate. If possible, data types of the source PSCS should be mapped to either xsd:integer, xsd:float, or xsd:string.

3.3 Class-specific Property Lists and Value Recommendations

As explained in section 2, most PSCS recommend the usage of certain properties for specific classes. In eCl@ss, this is a very strong component, as the set of recommended properties tries to reflect industry-wide consensus on which properties are necessary and sufficient to describe a product instance or model. Additionally, there is often a mapping between properties and enumerated values.

Both relations are worth being represented in the target ontology but do not fit into the open world assumption (OWA). It is not possible to restrict the usage of properties on certain classes or to restrict the usage of value instances to certain object properties, since the domain and range constructs in OWL and RDF-S are used to infer class membership instead of constraining allowed usage.

The most straightforward approach to deal with this problem is creating two annotation properties "recommendedProperty" and "recommendedValue" and use those to capture the respective relation. It is then up to the application developer to use this mapping to validate instance data or to help the user finding suitable properties or values for a given annotation task.

3.4 Keywords

Keywords providing alternate search paths to classes, properties, or values can be easily captured using the Dublin Core field "subject" as an annotation property dc:subject.

3.5 Versioning

As there is a high versioning dynamics in most PSCS, setting up a proper versioning mechanism is a core task of ontology engineering. Without capturing the underlying version of the respective PSCS, the resulting ontology is of very limited use.

Our current solution to ontology versioning for products and services ontologies that are derived from PSCS is as follows:

- We treat each new release of the ontology as a new ontology with new concepts, assuming no ex ante links between concepts in the old and new ontology.
- For classes, properties, and values that are known to be equivalent (e.g. based on an unchanged primary key and/or internal version ID), we provide a (huge) set of explicit "owl:sameAs", "owl:equivalentProperty", and "owl:equivalentClass" statements.

There are two motivations for this approach:

- One can find many “slight” changes in meaning between the same category in two PSCS versions, especially caused by more precise (narrowing down the concept) or more generic labels (broadening the concept).
- The notion of identity (e.g. the desired conceptual specificity) might vary between users of the same ontology or between fields of application. When it comes to aggregating invoice items for spend analysis, precision is not so much an issue, whereas for price comparison, we need a high specificity. With our approach, one can import different sets of identity assertions for different purposes.

The underlying rationale is that it seems much more important to not falsely assume identity than not to realize identity, for the later can be bridged later by mediation or established by explicit statements. This problem can be understood a trade-off decision between type 1 (false rejection) and type 2 (false acceptance) errors, or the calibration between precision and recall in Information Retrieval.

4 TRANSFORMING ECL@SS INTO AN OWL ONTOLOGY

Based on the methodology described above, we created an OWL Lite ontology of eCl@ss 5.0SP1. As eCl@ss is copyrighted material, we are currently preparing the legal framework for a public release of this ontology. Updated information will be made available at http://www.heppnetz.de/ont/eclass/.

4.1 Experiences

The eCl@ss standard is available at http://www.eclass.de in the form of separate CSV files containing categories, properties, values, class-property recommendations, property-value recommendations, and keywords. As the conversion process requires iterative queries, we first imported those files into a relational database. The actual conversion was then done by a Java application, accessing the database using JDBC.

The application of our methodology to eCl@ss creates only minor problems:

- The rich data type definitions had to be mapped to the three XSD data types currently supported by reasoners, i.e. xsd:integer, xsd:float, and xsd:string. Generally spoken are the original data type definitions more restrictive. Thus any legal eCl@ss data can be properly represented in the OWL variant. Precision issues could result when OWL data has to be converted back to fit into the original eCl@ss specification. We mapped all data types of the kind NR1* to xsd:integer, NR2* and NR3* to xsd:float, and all others to xsd:string.
- The strings, especially labels and definitions, contain special characters, like the ampersand (&), apostrophe (‘), and quotation mark (“), which are not allowed within XML values. To make the RDF/XML serialization of our ontology valid XML, we had to make sure that those characters are translated into proper XML codings (e.g. &amp; for the ampersand).
- The labels of enumerated values are often not self-contained. The meaning can sometimes only be grasped by looking at the recommended usage, i.e. for which property they are meant. For example, the value “WPA173003” is labeled “right”, but really means the door stop design type “right”.
- The resulting ontology is very big: About 25,000 categories in the source taxonomy result in more than 75,000 OWL classes, plus about 3,700 properties. Even though the current English version of eCl@ss does not contain long full text definitions of the classes, the total ontology size in RDF/XML exceeds 25 MB. This excludes the property and value recommendations, which we store in two separate OWL files, because they are not always needed and can be easily imported on demand.

The size of the ontology imposes unexpected problems when trying to use standard ontology editors (e.g. Protégé), repositories/APIs (e.g. Jena 2), or validators (e.g. vowlidator). They all exit with error messages when trying to process the full ontology. It was possible, though, to validate and use a restricted version of the ontology that contains only a small subset of the actual eCl@ss concepts.

4.2 Usage in e-Business Scenarios.

Product Description in the Semantic Web: We assume that "Fendt Supermower" is an agricultural machine (eCl@ss category AKK255001), its weight is 125.5 kg, and the manufacturer name is "Fendt". Assumed that the ID for this product instance is "machine1", the respective product description using the eCl@ss ontology would be as follows:

```
<pcs:C_AKK255001 rdf:ID="machine1">
 <pcs:P_AAA042001>125.50</pcs:P_AAA042001>  <!-- Weight -->
 <pcs:P_AAA001001>Fendt</pcs:P_AAA001001>  <!-- Manufacturer -->
 <pcs:P_AAA003001>Fendt Supermower1234</pcs:P_AAA003001> <!-- Name -->
</pcs:C_AKK255001>
```

Now, we want to search for all agricultural machines in the ontology that weigh less than 160 kg. The respective RDQL query would be:

```
SELECT ?x, ?weight, ?productName, ?vendor WHERE
(?x, <rdf:type>, <pcs:C_AKK255001-gen>)
(?x, <pcs:P_AAA001001>, ?vendor)
(?x, <pcs:P_AAA003001>, ?productName)
(?x, <pcs:P_AAA042001>, ?weight)
AND ?weight <160
```

Because we want to get only instances of the generic product category, the class to be used in the query is C_AKK255001-**gen**, not C_AKK255001-**tax**. The later could be used to determine all products that fall in the respective taxonomy category. For example, a store manager might want to see all products in this product segment, including maintenance and spare parts for agricultural machines. Just changing the class ID in the query to C_AKK255001-**tax** would return exactly that.

Annotation of Incoming Invoices for Spend Analysis: As described, the usage of this ontology is not limited to product description. It can also be employed to tag incoming invoices for spend analysis and cost accounting. For this, we need the additional class „IncomingInvoice" (a concept for paid invoices), the annotation property „costAccountingCategory", and the data type property „totalInUSD" for the total in US dollar. Then, we can annotate the incoming invoice over $ 1,200 for the above mentioned mower:

```
<pcs:IncomingInvoice rdf:ID="invoice1">
<pcs:totalInUSD>1200.00</pcs:totalInUSD>
<pcs:costAccountingCategory rdf:resource="&pcs;C_AKK255001"/>
</pcs:IncomingInvoice>
```

In order to find all incoming invoices related to the cost accounting category C_AKJ644001-tax ("Machine, device (for special applications)", which is a superordinate class to C_AKK255001-tax) and its subclasses, we can use the following RDQL query:

```
SELECT ?x, ?total WHERE
(?x, <rdf:type> <pcs:IncomingInvoice>)
(?x, <pcs:costAccountingCategory>, ?y)
(?y, <rdfs:subClassOf>  <pcs:C_AKJ644001-tax>)
(?x, <pcs:totalInUSD>, ?total)
```

Especially the fact that we have a sophisticated set of properties in the ontology allows for rich descriptions of the items, which eases rule-based content integration significantly. For example, we can use *a combination* of (1) taxonomy class information and (2) property ranges to automatically find the proper cost accounting category. A realistic scenario is that we infer the cost accounting ledger from a combination of the supplier name and the products and service category. For example, invoices referring to the category "Services (unclassified)" will be treated as "IT services" if the supplier is "IBM", and "Building maintenance" if the supplier is "Southwest Carpet Cleaning".

Accessing Recommended Properties and Property Values: We can also easily determine the recommended properties for a given class or the recommended property value instances for a given property with a simple RDQL query. As per definition, the properties are assigned to the annotation class, and not to the generic or taxonomy concept.

Find recommended properties for C_AKJ657001:

```
SELECT ?property WHERE
(<pcs:C_AKJ657001>, <pcs:recommendedProperty> ?property)
```

Find recommended values for ObjectProperty P_XYZ001001:

```
SELECT ?value WHERE
(<pcs:P_XYZ001001>, <pcs:recommendedValue> ?value)
```

5 CONCLUSION

In this paper, we introduced a methodology for deriving fully-fledged OWL Lite ontologies from products and services categorization standards. We showed how the intension of the taxonomy concepts is directly influenced by the interpretation of the taxonomic relationship, and how the creation of a set of three separate ontology classes representing (1) the generic concept, (2) the taxonomy concept, and (3) an annotation concept for each taxonomy category yields far more useful ontologies than one single ontology class per taxonomy category, especially if the later is in combination with rigid mapping of the taxonomy relationship to rdfs:subClassOf, as currently in use. Additionally, we developed an approach of how property dictionaries, enumerated values, class-property recommendations, and property-value recommendations can be captured in an OWL ontology while staying well within the limitations of OWL Lite.

We then successfully applied this methodology to the semantically rich categorization standard eCl@ss and yielded a comprehensive ontology for the product and services domain, reflecting more than 25,000 product types in 75,000 ontology classes plus a collection of 3,700 sophisticated properties. The resulting OWL version of eCl@ss 5.0SP1 is currently the most comprehensive products and services ontology that we know of, leaves the current "toy" level of other ontologies behind, and is fully usable for product description and spend analysis item tagging, which we regard as two very important usages of the Semantic Web.

It must be said, though, that the resulting ontologies are very big, with file sizes of more than 25 MB. This causes a problem with most current OWL tools and repositories, resulting in memory errors or very slow processing. The Jena framework and many Jena-based tools (e.g. vowlidator and Protégé 3.0) do not load an ontology of this size, but exit with an out-of-memory exception. We are currently trying to solve this by changing the memory allocation for the JVM. This might seem as an argument against the usefulness of such huge ontologies. For three reasons reasons, however, we think that quite the opposite is true: First, we must stress that ontologies of this size will rather be the lower limit of what we will have to expect for non-toy ontologies in e-Business scenarios. Using traditional relational databases and COTS application software, even SMEs are dealing with such amounts of data. Second, we conclude from the experienced limitations with current tools not that the ontology is too big, but that many prototypes yielded by the Semantic Web community have so far not achieved scalability beyond toy applications. Third, we expect our ontology to become a widely used benchmark among the developers of repositories and tools, and will thus hopefully contribute to a quick gain in scalability of Semantic Web software.

Also, readers from the ontology engineering field might criticize the shallowness of the generic concepts. It is true that the generic products and services concepts do not support much reasoning, for they are just named classes. However, the semantic richness needed for most business scenarios will come from the usage of the huge collection of properties. An example is a parametric search like "find all TV sets made by Siemens with a screen size between 10 and 15 inches and 12 Volt power-supply". Of course, we agree that a greater amount of axioms would be generally desirable. On the other hand, we see no simple way of automatically adding these axioms, because it can not be derived from the input standard. Additionally, we will have to put the resources necessary for the respective axiomatic enrichment in relation to the gain in automation and the resulting economies. In a domain as dynamic as products and services it should not be taken for granted that the business benefit will always outweigh the cost of the creation of the respective ontologies, nor that we are able to yield axiomatic richness during the lifespan of the concept.

Acknowledgements: I would like to thank Jos de Bruijn and Axel Polleres for very valuable comments.

REFERENCES

Berners-Lee, T. (1998). Cool URIs don't change. Available at: http://www.w3.org/Provider/Style/URI.html (Retrieved 08.11.2004).

Brachman, R. J. (1983). What IS-A Is and Isn't: An Analysis of Taxonomic Links in Semantic Networks. IEEE Computer 16 (10), 30-36.

de Bruijn, J. (2003). Using Ontologies. Enabling Knowledge Sharing and Reuse on the Semantic Web. DERI Technical Report DERI-2003-10-29, October 2003, 1-49.

Fairchild, A. M. and B. de Vuyst (2002). Coding Standards Benefiting Product and Service Information in E-Commerce. Proceedings of the 35th Annual Hawaii International Conference on System Sciences (HICSS-35), 258b.

Fensel, D., Y. Ding, B. Omelayenko, E. Schulten, G. Botquin, M. Brown and A. Flett (2001a). Product Data Integration in B2B E-Commerce. IEEE Intelligent Systems 16 (4), 54-59.

Fensel, D., D. L. McGuinness, E. Schulten, W. K. Ng, E.-P. Lim and G. Yan (2001b). Ontologies and Electronic Commerce. IEEE Intelligent Systems 16 (1), 8-14.

Fernández-López, M. and A. Gómez-Pérez (2002). Overview and analysis of methodologies for building ontologies. The Knowledge Engineering Review 17 (2), 129-156.

Hepp, M., J. Leukel and V. Schmitz (2004). Content Metrics for Products and Services Categorization Standards. Working Paper.

Kashyap, V. and A. Borgida (2003). Representing the UMLS Semantic Network using OWL. Proceedings of the 2nd International Semantic Web Conference 2003 (ISWC 2003), Sanibel Island, Florida, USA.

Kim, D., J. Kim and S.-g. Lee (2002). Catalog Integration for Electronic Commerce through Category-Hierarchy Merging Technique. Proceedings of the 12th International Workshop on Research Issues in Data Engineering: Engineering E-Commerce/E-Business Systems (RIDE'02), 28-33.

Kim, D., S.-g. Lee, J. Chun and J. Lee (2004). A Semantic Classification Model for e-Catalogs. Proceedings of the IEEE International Conference on E-Commerce Technology.

Klein, M. (2002). DAML+OIL and RDF Schema representation of UNSPSC. Available at: http://www.cs.vu.nl/~mcaklein/unspsc/ (Retrieved 23.04.2004).

McGuinness, D. L. (2001). UNSPSC Ontology in DAML+OIL. Available at: www.ksl.stanford.edu/projects/DAML/UNSPSC.daml (Retrieved 05.11.2004).

Ng, W. K., G. Yan and E.-P. Lim (2000). Heterogeneous Product Description in Electronic Commerce. ACM SIGEcom Exchanges 1 (1), 7-13.

Obrst, L., R. E. Wray and H. Liu (2001). Ontological Engineering for B2B E-Commerce. International Conference on Formal Ontology in Information Systems (FOIS'01), Ogunquit, Maine, USA, ACM.

Rector, A. L., Wroe, C., Rogers, J. and A. Roberts (2001). Untangling Taxonomies and Relationships: Personal and Practical Problems in Loosely Coupled Development of Large Ontologies. Proceedings of K-CAP'01, Victoria, British Columbia, Canada, ACM Press.

Schulten, E., H. Akkermans, G. Botquin, M. Dörr, N. Guarino, N. Lopes and N. Sadeh (2001). The E-Commerce Product Classification Challenge. IEEE Intelligent Systems 16 (4), 86-89.

U.S. Census Bureau (2004). North American Industry Classification System (NAICS). Available at: http://www.census.gov/epcd/www/naics.html (Retrieved 5.11.2004).

van Assem, M., M. R. Menken, G. Schreiber, J. Wielemaker and B. J. Wielinga (2004). A Method for Converting Thesauri to RDF/OWL. Accepted for publication, ISWC'04, Hiroshima, Japan, 1-15.

W3C (2001). XML Schema Part 2: Datatypes Second Edition. W3C Recommendation 28 October 2004. Available at: http://www.w3.org/TR/xmlschema-2/ (Retrieved 15.11.2004).

Wielinga, B. J., A. T. Schreiber and J. A. C. Sandberg (2001). From Thesaurus to Ontology. First International Conference on Knowledge Capture (K-CAP 2001), Victoria, British Columbia, Canada, ACM.

Wielinga, B. J., J. Wielemaker, G. Schreiber and M. van Assem (2004). Methods for Porting Resources to the Semantic Web,. Proceedings of the First European Semantic Web Symposium (ESWS'04), Heraklion, Greece, 299-311.