Extraction Process Specification for Materialized Ontology Views

Carlo Wouters[1], Tharam S. Dillon[2], Wenny Rahayu[1], Robert Meersman[3], and Elizabeth Chang[2]

[1] Department of Computer Science and Computer Engineering
La Trobe University, Bundoora, Victoria 3086, Australia
{cewouter,wenny}@cs.latrobe.edu.au
[2] Digital Ecosystems and Business Intelligence Institute,
Curtin University of Technology,
Perth, Australia
tharam.dillon@cbs.curtin.edu.au,
elizabeth.chang@cbs.curtin.edu.au
[3] STARLab, Department of Computer Science
Vrije Universiteit Brussel, Brussel, 1050, Belgium
Robert.Meersman@vub.ac.be

Abstract. The success of the semantic web relies heavily on ontologies. However, using ontologies for this specific area poses a number of new problems. One of these problems, extracting a high quality ontology from a given base ontology, is currently receiving increasing attention. Areas such as versioning, distribution and maintenance of ontologies often involve this problem. Here, a formalism is presented that enables grouping ontology extraction requirements into different categories, called optimization schemes. These optimization schemes provide a way to introduce quality in the extraction process. An overview of the formalism is discussed, as well as a demonstration of several example optimization schemes. Each of these optimization schemes meets a certain requirement, and consists of rules and algorithms. Examples of how the formalism is deployed to reach a high-quality result, called a materialized ontology view, are covered. The presented methodology provides a foundation for further developments, and shows the possibility of obtaining usable ontologies in a highly automated way.

ACM Subject Descriptors ('98): H.3.5 [INFORMATION STORAGE AND RETRIEVAL]: Web-based services; I.1.2 [SYMBOLIC AND ALGEBRAIC MANIPULATION]: Algorithms; I.2.4[ARTIFICIAL INTELLIGENCE]: Semantic networks --- Representation languages.

Additional Keywords: Ontology Extraction.

1 Motivation

In recent years, the unstructured storage of data, especially on the World Wide Web, and the difficulties experienced with retrieving relevant data with the existing search engines, have triggered new research aimed at ameliorating information retrieval and storage. New ways of storing information meant for the Internet were developed, such as XML [W3C 1999], HTML[a] [Fensel, Decker et al. 1998], DTD and RDF. These languages provide a tool to store the information in a structured way, but with that another problem arose; everyone was free to develop there own taxonomy of how they want to categorize their information, e.g. [Heflin, Hendler et al. 1999; Van Harmelen and Fensel 1999]. It is clear that widely accepted standards should be used as metadata to define how the actual information is split up, no matter what language or syntax is used to implement this. These widespread standards are formulated as ontologies.

The first wave of ontology applications and researchers mainly concentrated on getting an effective system up, solving the apparent issues that had been holding back knowledge acquisition from the Internet and related resources. A number of these have turned out to be beneficial, without any of them clearly standing out, and no single standard has been agreed upon [Hovy 1998]. Since then we have seen merging of some of the standards – e.g. OIL incorporating elements of OKBC, XOL and RDF [Fensel, Horrocks et al. 2000], Ontolingua using KIF [Genesereth 1991; Genesereth and Fikes 1992; Gruber 1992], DAML and OIL into DAML-OIL [Berners-Lee and Al 2001] (currently being reworked into OWL [W3C 2002c]) – and diversification of others.

Now that the first generation of ontology applications has settled in, more complicated issues and considerations have reared their heads, such as the quality of ontologies in all its facets [Colomb and Weber 1998; Hahn and Schnattinger 1998; Kaplan 2001; Guarino and Welty 2002; Holsapple and Joshi 2002; Wouters, Dillon et al. 2002b]. Improvements need to be made to the systems that are already in place, and theoretical and practical modifications have to be made to cater for versioning, maintenance and distribution of ontologies [Klein, Fensel et al. 2002]. Furthermore, a further integration of different existing systems is needed [McGuinness, Fikes et al. 200; Noy, Sintek et al. 2001].

The ontologies used also tend to grow larger, to a point where ideally the entire world is modeled in one super-ontology (through the use of upper ontologies [Lenat 1995]), providing great compatibility and consistency across all sub-domains, but practically it introduces the new problem of being too vast to be used in its entirety by any application. Considering the internet as a data repository, it seems clear that users with a very slow or costly connection to this repository might opt to get a local, modified copy of the repository to base views upon, and to query in other ways. It seems highly unlikely that someone will be able to copy the entire contents of the World Wide Web to a local repository, and even more unlikely that all this data will be actually used in whatever application the user might intend it to be used for. If a business just needs access to information on the share market, it would not benefit from all the other information that their local copy would contain. This is just one of the many reasons why a complete ontology might not be a valid structuring option for certain users. Another reason can be found in varying levels of security and confidentiality – not necessarily every user of an ontology has the same access rights, and using a smaller ontology, merely containing the appropriate parts of the base ontology might enable local copies. Efficiency of querying repositories might be another reason for having a simplified, local version of an ontology, and there are many more.

It is imperative that when an ontology view is derived, the quality of the resulting ontology is as high as possible. First of all, the intentions of the ontology engineer should obviously be satisfied, and the resulting design should be a consistent, cohesive, complete and well-formed ontology. Secondly, the result should be further pruned, i.e. is the obtained solution one of the most efficient, flexible, simple, versatile, etc. solutions (varying with the specific needs of the ontology engineer).

The resulting ontology should be usable as the base for an independent system, i.e. be an ontology in its own right. This article aims to identify the processes involved in this extraction. It is set up in such a way that the theoretical definitions and processes can also be readily transformed and applied to other new research, such as versioning and distribution of ontologies [Wouters, Dillon et al. 2002a]. While establishing the details of the sub-processes used for the extraction, great care is taken to obtain a result that adheres to the quality assurance issues raised before. Finally, a brief outline of future research will be given, eventually leading up to an integrated transformation environment for ontologies, interfacing with existing international standards.

2 Defining the Ontology

Although different standards have emerged, and none of them has achieved the status of the ultimate ontology definition, this does not pose a huge problem. The use of wrappers, which can be used to extract data – and semantic information – from a certain type of ontology definition and implementation, and translate it to another, helps with this. Another way in which wrappers can be used is in trying to transform the normal natural language, textual information resources on the web to more structured resources [Muslea 1999].

None of these approaches however, provides a consistent, reliable solution towards the future. There is a clear need to steer towards a more solid foundation in terms of theoretical definitions and international standards.

2.1 Overview of Ontology Definitions

A problem that occurs when doing any research in the field of ontologies is that no single definition for an ontology exists that can be completely utilized in Information Technology. Definitions have been modified, and diversified over the last couple of years, but the problems remain; either a proposed definition is too general to have any practical value [Gruber 1993; Fensel 2001], or the definition provides enough preciseness, but only suits a small number of models and systems in use today [Noy and Hafner 1997]. In practical applications this sometimes is overcome by introducing several 'layers' for ontologies, for example the ontological commitments in [Spyns, Meersman et al. 2002]. It is not within the scope of this article to produce a final solution to this problem, but there is still the necessity for clarifying what definitions will be used throughout this article. The initial definition will be extended as new definitions are introduced, incorporating stricter requirements for an ontology.

The first definition clarifies what cardinalities are used. Please refer to Appendix A for the multiplication table of these cardinalities.

Definition 1

$$\textit{The set of cardinalities card} \equiv \{0,\ 1,\ m\} \qquad (1)$$

Definition 2

$$\vartheta \equiv \{\text{The set of all ontologies}\} \qquad (2)$$

Definition 3. Let an ontology O be a six tuple (<C, A, attr, B, M_a, M_b>),consisting of a non-empty, finite set of concepts C, a finite set of attributes A, a mapping attr, a set of binary relationships B, an attribute cardinality mapping M_a, and a relationship mapping M_b, with attr the mapping of concepts onto elements of A, B the disjoint union of the binary associative relationships B_s, the binary inheritance relationships B_i, and the binary aggregate relationships B_{agg}.

$$O \in \vartheta \Leftrightarrow O \equiv < C, A, attr, B, M_a, M_b > \qquad (3)$$

with

$$C = finite \wedge C \neq \varnothing$$

$$A = finite$$

$$attr: C \rightarrow A$$

$$B \subseteq C \times C \wedge B = B_s \cup B_i \cup B_{agg}$$

$$M_a: attr \rightarrow card^2$$

$$M_b: B \rightarrow card^4$$

Intuitively, ontologies conceptually represent a perceived world through concepts, attributes and relationships. Concepts may represent the different higher-level components of this world, and each component may have attributes. These attributes may be derived from the characteristics of components of the world. Relationships may also hold between these concepts.

For practical reasons, only binary relationships are considered here. Note that binary relationships occur most frequently in current modeling. Unary models are not modeled, since these are taken care of by forming subtypes. N-ary relationships are not considered as transformation of this type of relationship to binary relationships is possible. For a more comprehensive treatment of arity of relationships see [Halpin 1995].

Some useful notations are given here to enhance readability throughout the rest of the article.

Notation 1

Given an ontology $O = < C, A, attr, B, M_a, M_b >$ and an attribute mapping $t = (c,a)$, with $c \in C$, $a \in A$ (4)

we denote

$$\pi_1(t) = c,$$

$$\pi_2(t) = a,$$

$$attr(c) = \{a \in A \mid \exists t \in attr : t = (c,a)\}$$

Notation 2

Given an ontology $O=<C, A, attr, B, M_a, M_b>$ and a binary relationship $b=(c_1, c_2)$, with $c_1, c_2 \in C$ (5)

we denote

$\pi_1(b)=c_1$

$\pi_2(b)=c_2$

Notation 3

Given an ontology $O=<C, A, attr, B, M_a, M_b>$ and (6)

An attribute cardinality $m=(n_1, n_2)$ for an attribute mapping $t \in attr$

with $n_1, n_2 \in card$, and t the attribute mapping for attribute a

we denote

$m(t)=m(a)=(n_1, n_2)$

$m_{min}(t)=m_{min}(a)=n_1$

$m_{max}(t)=m_{max}(a)=n_2$

Notation 4

Given an ontology $O=<C, A, attr, B, M_a, M_b>$ and (7)

a relationship cardinality $m=(n_1, n_2, n_3, n_4)$ for a binary relationship b

with $n_1, n_2, n_3, n_4 \in card$

we denote

$m(b)=(n_1, n_2, n_3, n_4)$

$m_{min1}(b)=n_1$

$m_{max1}(b)=n_2$

$m_{min2}(b)=n_3$

$m_{max2}(b)=n_4$

Figure 1 gives an example of a UML diagram, and how this can be interpreted using the notations defined above, is shown in Table 1.

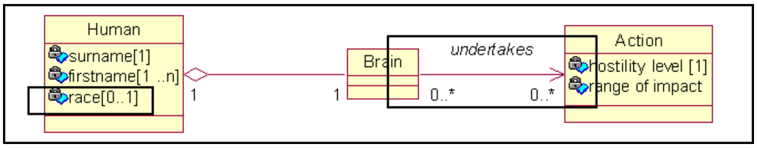


Fig. 1. Example UML Diagram

Table 1. Mapping from UML Diagram to Defined Notation

Statement in UML	Notation
Race is attribute	Race $\in$ A
Race has cardinality [0..1]	m(race) = (0,1) m_{min}(race) = 0 m_{max}(race) = 1
'undertakes' is relation-ship	undertakes $\in$ B
Brain and Action are the connected concepts	undertakes = (Brain, Action) π_1(undertakes) = Brain π_2(undertakes) = Action
undertakes has cardinal-ities 0..* and 0..*	m(undertakes) = (0, n, 0, n) m_{min1}(undertakes) = 0 m_{max1}(undertakes) = 1 m_{min2}(undertakes) = 0 m_{max2}(undertakes) = 1

Obviously, the minimum cardinality for an attribute mapping should never be more than the maximum cardinality, so an additional rule always applies:

Transformation Rule 1

$$\textit{Given an ontology } O=<C, A, attr, B, M_a, M_b> \tag{8}$$

$$\forall m \in M_a : m_{min}(t) \leq m_{max}(t), \textit{ with } t \in attr$$

$$\forall m \in M_b : m_{min1}(t) \leq m_{max1}(t), \textit{ with } t \in attr$$

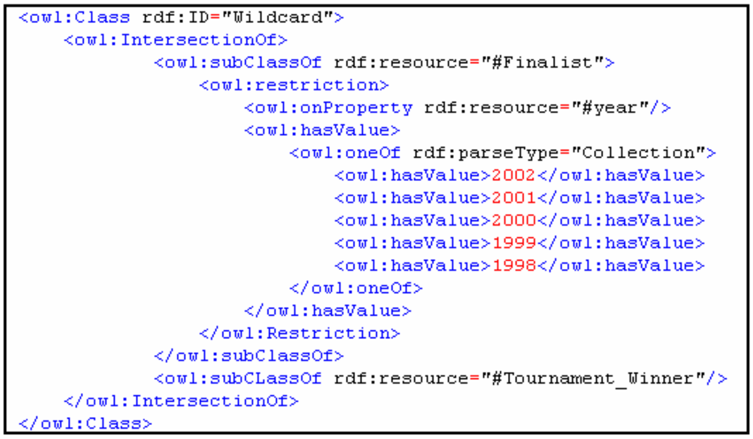
$$\forall m \in M_b : m_{min2}(t) \leq m_{max2}(t), \textit{ with } t \in attr$$

Following [Spyns, Meersman et al. 2002] the 'semantics' of an ontology is the range of interpretation mapping of an application environment onto an ontology. Note that the semantics of the real world problem are replaced by an ontology. Some examples of an application environment are RDBMS, software applications, documents, web-site, etc. Whilst it is recognized that there are important differences between an ontology, and a conceptual model, for the purpose of deriving sub-ontologies these are immaterial. Frequently, a conceptual model can be considered to be an ontology expressed in a chosen syntax. However, this syntax should not impact the definitions and theorems presented, so for the purpose of this paper this difference is irrelevant.

Throughout this paper UML [Rumbaugh, Jacobson et al. 1999] is sometimes used to graphically represent an ontology, but it is not the intention to show the suitability of UML for the modeling of ontologies. UML is merely a convenient modeling notation that for practical reasons was chosen to highlight aspects of ontologies. There should be no confusion as to the difference of the ontology and the modeling notation used to represent it, and by no means is it the intention of the authors to promote this modeling notation to a higher status. UML is used to represent object oriented models, and as our definition of an ontology (section 2) has concepts (similar to classes), attributes and relationships, it was found convenient to use this easy to read data model to illustrate aspects of semantics of ontologies. One could, however, choose any alternative notation such as semantic nets [Feng, Chang et al. 2002] to illustrate the issues. This does not distract from the methodologies in this paper.

2.2 Concepts in Detail

What is referred to as ‘concept’ in the definition is a very broad ontological element. It can be defined in a variety of ways, potentially providing a lot of additional information about itself, and its relationships (and topological proximity) to other elements. Through an inheritance structure concepts can be made specializations and generalizations of other concepts. The conventions used for a ‘concept’ are exactly the same as what is labeled a ‘class’ in OWL [W3C 2002c]. Instead of ‘definition’ of a concept, such declarations of a concept are called ‘axioms’[1]. An example of an axiom defining a concept (labeled ‘owl:class’) is given in Figure 2, where a “Wildcard” is axiomatically defined as the result of Boolean operations on subsets of other classes.



```
<owl:Class rdf:ID="Wildcard">
    <owl:IntersectionOf>
            <owl:subClassOf rdf:resource="#Finalist">
                <owl:restriction>
                    <owl:onProperty rdf:resource="#year"/>
                    <owl:hasValue>
                        <owl:oneOf rdf:parseType="Collection">
                            <owl:hasValue>2002</owl:hasValue>
                            <owl:hasValue>2001</owl:hasValue>
                            <owl:hasValue>2000</owl:hasValue>
                            <owl:hasValue>1999</owl:hasValue>
                            <owl:hasValue>1998</owl:hasValue>
                        </owl:oneOf>
                    </owl:hasValue>
                </owl:Restriction>
            </owl:subClassOf>
            <owl:subCLassOf rdf:resource="#Tournament_Winner"/>
    </owl:IntersectionOf>
</owl:Class>
```

Fig. 2. Example OWL statements in RDF for axioms[2] [W3C 2002b]

Note in Figure 2 that a restriction results in a concept, which is here unnamed, as it is only used as a mechanism to define another (labeled) concept.

2.3 Attributes and Relationships in Detail

Both Attributes and Relationships can be regarded as properties that belong to a concept. In fact, in [W3C 2002a; W3C 2002b] the naming convention used is indeed “Property” for both these terms. However, it is not uncommon to make a further

[1] Note that throughout the rest of this article the term ‘axiom’ is used as specified in the OWL definition W3C (2002c). OWL Web Ontology Language 1.0 Reference. *W3C Working Draft.* However, readers with a background in A.I. and mathematics should be aware of the broader meaning of this term here, otherwise it could lead to erroneous interpretation of the definitions and theorems presented.

[2] As the final syntax of OWL still has to be finalized, this example should be taken as an indication of a possible representation, rather than an exact OWL RDF file example.

distinction between the two types of properties. For example, [W3C 2002a] groups the properties in "DatatypeProperties" (i.e. attributes) and "ObjectProperties" (i.e. relationships). Because of the distinction in semantic meaning between these two types of properties, the names "Attribute" and "Relationship" were preferred, as they clearly indicate the semantic meaning, and relate closer to the naming convention as used in Object Oriented Model, and Relational Database Management Schemas.

2.4 Restrictions to Obtain Practical Ontology Definition

The formal definition as proposed above is at too high a level of abstraction to be practically usable. To make the ontology definition more practical, a couple of refinements are introduced.

The first restriction that is made, which is an integral part of the ontology definition is the fact that an attribute that is a part of an ontology should at least belong to one concept. This is given in the following extension to the definition of an ontology.

Definition 4

$$\forall O \in \vartheta : O = < C, A, attr, B, M_a, M_b > \Rightarrow \forall a \in A, \exists c \in C : a \in attr(c) \quad (9)$$

This definition extension ensures that there is always a semantic meaning of the attribute in relation to the rest of the ontology. Note that some standards do not have this restriction (e.g. OWL [W3C 2002c]). Here it is important to regard the ontology as semantically interlinked, as this provides a solid lower boundary for algorithms.

Furthermore, the boundaries of a single ontology need to be established, i.e. when does one ontology cease to be one ontology, and become multiple ontologies. The following definitions aim to support the notion of a well-formed ontology.

Definition 5. Let a graph G consist of a set of vertices V, and a set of edges E, with an edge defined by two vertices in V.

$$G \equiv < V, E > \quad (10)$$

with

$$V = finite \wedge V \neq \varnothing$$

$$E \subseteq V \times V$$

This is just a standard definition of a graph [Biggs, Lloyd et al. 1976], and is merely given to facilitate the consequent definitions and theorems. For more information on Graph Theory, a number of sources are readily available (e.g. [Von Staudt 1847; Biggs, Lloyd et al. 1976]).

Definition 6

Given a graph $G = < V, E >$ (11)

we define

G' *an island in* $G \Leftrightarrow G' \equiv < V', E' >$

with

$V' \neq \emptyset$

$V' \subseteq V$

$E' \subseteq E$

$G' \neq G$

$\forall v \in V', \ \forall v' \in V \setminus V' : (v, v') \notin E$

An island is another term for what is called 'proper component' of a graph [Biggs, Lloyd et al. 1976], with slight modification to suit our purposes. Next, an Ontology Graph is defined. Ontology graphs will aid in the more 'geographical' or 'topological' characteristics of an ontology, and their implications.

Definition 7

$$\textit{Given an ontology } O = \langle C, A, attr, B, M_a, M_b \rangle \quad (12)$$

We define

G_O *is an Ontology Graph for* $O \Leftrightarrow G_O \equiv \langle V_O, E_O \rangle$

with

$V_O = C$

$E_O = B$

G_O *has no islands*

It is important to note here that a valid Ontology Graph cannot contain an island. Through an extension to the definition of a (valid) ontology, this requirement for an Ontology Graph becomes a requirement for an ontology as well.

Definition 8

$$\forall O \in \vartheta \Rightarrow \exists \textit{ an Ontology Graph } G_O \textit{ for } O \quad (13)$$

This definition says that an ontology is not considered valid unless there is an Ontology Graph that can be associated with it. Because an Ontology Graph cannot have an island, this means that a valid ontology has to have a corresponding graph representation without islands.

For completeness, the full definition for an ontology is given here, i.e. the addition of Definition 4 and Definition 8 to Definition 3.

Definition 9

$$\textit{Given a set of concepts } C \quad (14)$$

We define

An ontology over C, $O \in \vartheta \Leftrightarrow O \equiv \langle C, A, attr, B, M_a, M_b \rangle$

with

$C = finite \wedge C \neq \emptyset$

$A = finite$

$Attr: C \rightarrow 2^A$

$B \subseteq C \times C \wedge B = B_r \cup B_i \cup B_{agg}$

$M_a: attr \rightarrow card^2$

$M_b: B \rightarrow card^4$

$\forall a \in A, \exists c \in C: a \in attr(c)$

$\exists$ *an Ontology Graph* G_O *for* O

2.5 Ontology Complement Theorem

In order to define a complement of an ontology for a given base ontology, first the notion of subsets needs to be defined. The following two definitions indicate a subtle but important difference; not all subsets are ontologies, the subsets that are ontologies in their own right are called sub-ontologies.

Definition 10

Given an ontology $O=<C, A, attr, B, M_a, M_b>$ (15)

we define

a six tuple S *is a subset of* $O \Leftrightarrow S \equiv < C_1, A_1, attr_1, B_1, M_{a1}, M_{b1}>$

with

$C_1 \subseteq C$

$A_1 \subseteq A$

$attr_1 \subseteq attr$

$B_1 \subseteq B$

$M_{a1} \subseteq M_a$

$M_{b1} \subseteq M_b$

Notation 5

Given an ontology $O=<C, A, attr, B, M_a, M_b>$ *and a subset* S *of* O (16)

we denote

$S \subseteq O$

The definition of a subset is expanded to incorporate the requirements for a valid ontology. Although 'a subset that is a valid ontology' is sufficient for a definition, we will expand the definitions into the individual requirements, and combine them into one more detailed definitions for a sub-ontology.

Definition 11

Given an ontology O (17)

we define

$$O' \text{ is a sub-ontology of } O \Leftrightarrow O' \in \vartheta \wedge O' \subseteq O$$

Notation 6

Given an ontology O and a sub-ontology O' of O (18)

we denote

$$O' \Subset O$$

Two very basic Ontologies, O_1 and O_2 are presented in Figure 3. The ontology to the right (O_2) is a sub-ontology of the ontology to the left (O_1), i.e. it is a valid ontology in its own right, as well as being a subset of O_1.

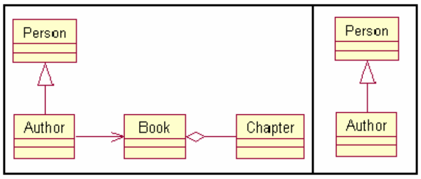


Fig. 3. Two simplified base ontologies (left: O_1, right: O_2)

Definition 12

Given ontologies $O_1=<C_1, A_1, attr_1, B_1, M_{a1}, M_{b1}>$, $O_2=<C_2, A_2, attr_2, B_2, M_{a2}, M_{b2}>$ (19)

with

$$O_2 \Subset O_1$$

We define

$$O_1 \backslash O_2 \equiv < C_1 \backslash C_2, A_1 \backslash A_2, attr_1 \backslash attr_2, B_1 \backslash B_2, M_{a1} \backslash M_{a2}, M_{b1} \backslash M_{b2} >$$

In this definition the characteristics of the complement of a sub-ontology are presented. An interesting issue is raised by this definition, namely is the complement ($O_1 \backslash O_2$) always a valid ontology, or not ? The answer to this question is negative, as following example illustrates a complement that is not a valid ontology.

Looking back at Figure 3, the first thing to note is that the basic requirement (antecedent of formula (19)) is fulfilled, and that O_2 is a sub-ontology of O_1 ($O_2 \Subset O_1$). Secondly, in this simplified form, O_1 counts 7 elements (or 7 that matter in the example), being 4 concepts and 3 relationships. On the other hand, O_2 only has 3; 2 concepts and 1 relationship.

Looking at the complement of O_2 ($O_1 \backslash O_2$), it can easily be visually verified in Figure 4 that there is an invalid relationship.

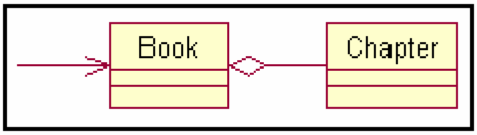


Fig. 4. The resulting graphical representation of set $o_1 \backslash o$

The relationship between 'Author' and 'Book' is an element of the binary relationship set of O_2. From the definition this set B_2 is a subset of $C_2 \times C_2$, but one of the concepts connected by the relationship is an element of C_1, and from this definition cannot be an element of C_2, hence $O_1 \backslash O_2$ is not an ontology.

The obvious question to answer now is when this complement would be an ontology. The following theorem provides the key to solving this problem.

Theorem

Given an ontology $O=<C, A, attr, B, M_a, M_b>$, and its corresponding ontology graph $G_O=<V, E>$ (with $V=C$, $E=B$) (20)

$$\forall O_1 \in \vartheta : O_1 \Subset O \Rightarrow O \backslash O_1 \notin \vartheta$$

Proof

1. O_1=O

$\Rightarrow O \backslash O_1 = O \backslash O = \varnothing$

$\wedge\ \varnothing \notin \vartheta$ ($C \neq \varnothing$ for valid ontology)

$\Rightarrow O \backslash O_1 \notin \vartheta$

2. $O_1 \neq$O

We give a proof by contradiction. Following statement is proven wrong:

$$\exists O_1 \in \vartheta : O_1 \Subset O \Rightarrow O \backslash O_1 \in \vartheta$$

Let $O_d = O \backslash O_1$

If $O_d \in \vartheta$ then, per definition (of ontology) there exists an ontology graph $G_d=<C \backslash C_1, B \backslash B_1>$. *Let* $V_d= C \backslash C_1$, $E_d= B \backslash B_1$

We now show that G_d is an island in G_O, which is not permitted (see section 2.4).

Indeed, let $v_d \in V_d$ and be $v \in V_1$ arbitrary

Clearly;

Either

$(v_d, v) \in Ed$	- not permitted because $v \in V_1 (=C_1)$, so $v \notin V_d (=C \backslash C_1)$
or $(v_d, v) \in E1$	- not permitted because $v_d \in V_d (=C \backslash C_1)$, so $v_d \notin V_1 (=C_1)$
or $(v_d, v) \notin E$	- Only remaining option ($E=E_d \cup E_1$), so must be this

In other words, there is no edge that can be found that has a vertex in V_d and one in V_1. This means that the two corresponding graphs have no connection, and are thus separate islands. Thus $G_O(=G_1 \cup G_d)$ itself would consist of two islands, and thus is in conflict with the definition of ontology graphs (section 2.4).

The above proof shows that it is true that the complement of an ontology and one of its sub-ontologies is never an ontology.

2.6 Materialized Ontology Views

The result of an extraction process is not just simply referred to as an extracted ontology, but rather an extracted materialized ontology view. This section will discuss the term 'materialized ontology view', and why it is used to describe the result of the extraction process.

In the extraction process, no new information should be introduced (e.g. adding a new concept). However, it is possible that existing semantics are represented in a different way (i.e. a different view is established). This is similar to the use in the database area, where views do exactly that [Chen 1976; Date 2000], and thus the notion of an ontology view can be obtained by an analogy to the database view definition, with some minor modifications.

Definition 13

A Materialized Ontology View of a base ontology is a (valid) ontology that consists solely of projections, copies, compressions, and/or combinations of elements of the base ontology, presenting a varying and/or restricting perception of the base ontology, without introducing new semantic data. (21)

Intuitively, the definition states that – starting from a base ontology – elements can be left out and/or combined at will, as long as the result is a valid ontology again. In the process, no new elements should be introduced (unless the new element is the combination of a number of original elements, i.e. the compression of other elements).

A *materialized* ontology view is required, as the resulting ontology should be an independent ontology, i.e. be a valid ontology, even if the base ontology is taken away. This requires a materialization of the result, as opposed to leaving it as virtual, which would result in a dependent view.

3 The Ontology Extraction Process

The main issues covered in the introduction concerning current research on ontology transformations, such as versioning, distribution, evolution, bear numerous similarities. For instance, an existing ontology (from now on referred to as "base ontology") is present and used as a starting point, and through some changes a new ontology needs to be established. It is vital to identify the different elements of this process, so that one can concentrate on the appropriate issues in the research which need to be resolved.

Naturally, there are certain requirements and constraints that a derived ontology or materialized view must abide by. One of the constraints that comes to mind as being of great importance in versioning and distribution is backward compatibility, or compatibility in general [Heflin and Hendler 2000; Klein and Fensel 2001; Kim 2002]. Some of the lower level requirements to serve purposes like compatibility are explored here. Before these can be explored it is necessary to introduce a labeling of an Ontology here.

3.1 Labeling an Ontology

In this section, a new six tuple <C', A', attr', B', M_a', M_b'> is constructed by applying a labeling to the base ontology, and then only using elements with certain labels. In other words, every element of the ontology receives a certain label, and this label determines whether it will be a part of the new six tuple or not. There are three main reasons for introducing such a labeling throughout the extraction process. Firstly, this labeling facilitates user manipulation of the extraction process, as it provides a way for the user to specify key elements.

Secondly, in the optimization step this labeling is also re-applied (modification of present labeling) by every optimization scheme. This is the standard way that different components of the extraction process can communicate with each other. Every optimization scheme is able to read in a labeling, incorporate it into its algorithms, and modify the labeling according those algorithms. The next optimization scheme in line uses this modified labeling as its input. This means that the changes one optimization scheme wants to make effectively are communicated to the next optimization scheme.

Thirdly, the labeling is also used to produce a final result. As indicated previously, this is done by constructing a new six tuple through the use of a label filter.

A few necessary definitions are given next, followed by the automated way of labeling the attribute mapping that is assumed in examples throughout this paper.

Definition 14

the selection set S is defined as (22)

$$S \equiv \{selected, deselected, void\}$$

From the previous definition it can be noted that there are three possible labels that are used; selected, deselected and void.

Definition 15

A labeling of an ontology O=<C, A, attr, B, M_a, M_b> is a mapping σ such that (23)

$$\sigma(O) = \sigma(<C, A, attr, B, M_a, M_b>) = <C', A', attr', B', M_a', M_b'>$$

with

$$C' = \sigma_C(C) = \{c \in C \mid \sigma_C(c) \neq \text{'deselected'}\},$$

$$A' = \sigma_A(A) = \{a \in A \mid \sigma_A(a) \neq \text{'deselected'}\},$$

$$attr' = \sigma_{attr}(attr) = \{t \in attr \mid \sigma_{attr}(t) \neq \text{'deselected'}\},$$

$$B' = \sigma_B(B) = \{b \in B \mid \sigma_B(b) \neq \text{'deselected'}\},$$

$$M_a' = \sigma_{Ma}(M_a) = M_a \text{ and } M_b' = \sigma_{Mb}(M_b) = M_b$$

This labeling is crucial in the interaction between humans and algorithms, and algorithms amongst themselves. An algorithm might optimize a solution set in a certain way, and needs to pass this on to the next algorithm. Often there are practical subjective influences that cannot be translated in an algorithm, and the labeling allows a user to express his/her wishes. For instance, certain information might not be made available to the materialized view, and thus by using this labeling system, this requirement can readily be communicated to the system. In this case, the unavailable elements would be deselected, so that for an undesirable ontological element e, σ(e)=deselected.

Some further definitions are needed to enable quick reference to a certain set of ontological elements. As the definitions for all the sets in the ontology six tuple are very similar, one generic definition is given (rather than six slightly varying ones):

Definition 16

$$\text{Given an ontology } O = <C, A, attr, B, M_a, M_b> \tag{24}$$

$$\sigma_X : X \rightarrow S$$

with

$$X \in \{C, A, attr, B, M_a, M_b\}$$

Next we demonstrate how the previous definitions are applied in the ontology extraction process. This is done by giving a small example. The labeling will reoccur in many examples throughout this paper, as it is an essential aid in the extraction process, and serves multiple purposes (as discussed previously).

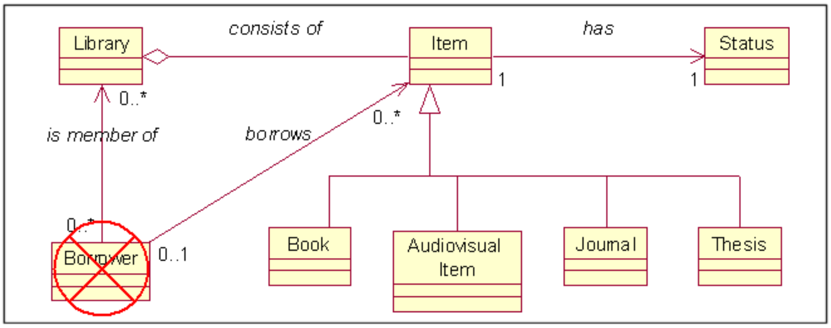


Fig. 5. UML Representation of University Example (Simplified)

When a university decides to develop a search system for its students, they use a certain library ontology as a basis, i.e. the library ontology is the base ontology for their materialized ontology view. The data they use comes from their own library which also uses the library ontology. However, the library notifies them that certain types of data will not be available, as that would infringe the privacy of the library members. Specifically, a concept named "Borrower" is off limits for the university. Although the university still wants the optimization algorithms to decide what a good ontology is to use, they now have to make this requirement clear to the algorithms, and the labeling does exactly that. By applying the labeling $\sigma_C(\text{"Borrower"})=deselected$ the algorithms have the appropriate information to further process the transformation.

3.1.1 Automation of attr Labeling

Although not necessary, it is convenient to use an automatic labeling of the *attr* mapping. Here we will examine all the possible combinations of the related concept-attribute pair, and give a desired labeling for the attribute relationship between them. Afterwards these results are summarized in a couple of rules.

given an ontology $O=<C, A, attr, B, M_a, M_b>$
$\forall t \in attr:$

1) $\sigma_C(\pi_1(t))= selected \wedge$
   a. $\sigma_A(\pi_2(t))= selected$
      $\Rightarrow \sigma_{attr}(t)= selected$
   b. $\sigma_A(\pi_2(t))= deselected$
      $\Rightarrow \sigma_{attr}(t)= deselected$
   c. $\sigma_A(\pi_2(t))= void$
      $\Rightarrow \sigma_{attr}(t)= void$
2) $\sigma_C(\pi_1(t))= deselected \wedge$
   a. $\sigma_A(\pi_2(t))= selected$
      $\Rightarrow \sigma_{attr}(t)= deselected$
   b. $\sigma_A(\pi_2(t))= deselected$
      $\Rightarrow \sigma_{attr}(t)= deselected$
   c. $\sigma_A(\pi_2(t))= void$
      $\Rightarrow \sigma_{attr}(t)= deselected$
3) $\sigma_C(\pi_1(t))= void \wedge$
   a. $\sigma_A(\pi_2(t))= selected$
      $\Rightarrow \sigma_{attr}(t)= void$
   b. $\sigma_A(\pi_2(t))= deselected$
      $\Rightarrow \sigma_{attr}(t)= deselected$
   c. $\sigma_A(\pi_2(t))= void$
      $\Rightarrow \sigma_{attr}(t)= void$

The previous list gives the desired automated mapping result for each individual case, and now we give a more comprehensive set of rules that have the same effect.

Attribute Mapping Automation Rule 1

$$\textit{given an ontology } O=<C, A, attr, B, M_a, M_b> \tag{25}$$

$$\forall t \in attr: \sigma_C(\pi_1(t))=selected \wedge \sigma_A(\pi_2(t))=selected \Leftrightarrow \sigma_{attr}(t)= selected$$

Attribute Mapping Automation Rule 2

$$\textit{given an ontology } O=<C, A, attr, B, M_a, M_b> \qquad (26)$$

$$\forall t \in attr: \sigma_C(\pi_1(t))= deselected \vee \sigma_A(\pi_2(t))= deselected \Leftrightarrow \sigma_{attr}(t)= deselected$$

Attribute Mapping Automation Rule 3

$$\textit{given an ontology } O=<C, A, attr, B, M_a, M_b> \qquad (27)$$

$$\forall t \in attr: (\sigma_C(\pi_1(t))= void \wedge \sigma_A(\pi_2(t)) \neq deselected) \vee (\sigma_C(\pi_1(t)) \neq deselected \wedge \sigma_A(\pi_2(t))=void) \Leftrightarrow \sigma_{attr}(t)= void$$

Note that the last rule follows directly from the two previous rules. As there are only three mapping possibilities, if the first two don't apply, it has to be the third mapping. Thus, with only the first two rules a full mapping can be established (if we assume a default 'void' labeling), but the third rule was added for reasons of completeness.

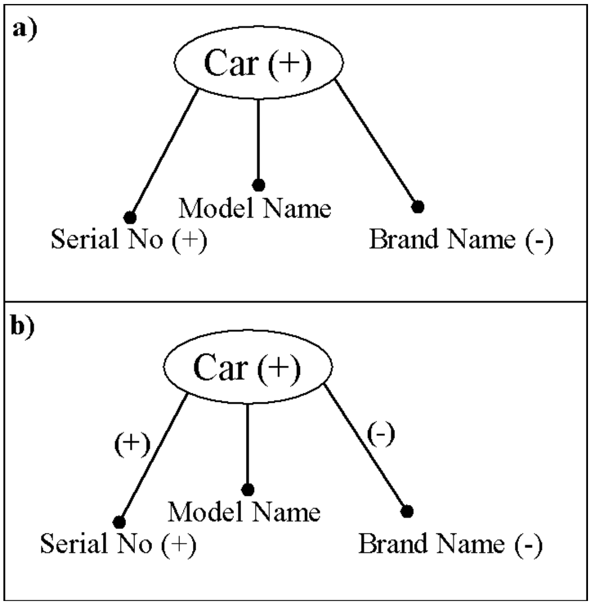


Fig. 6. Attribute Mapping before (a) and after (b) Automated Labeling

Some examples of these labeling rules are demonstrated in Figure 6. The concept "Car" is selected, and it has three attributes with different labels. Note that a missing indication means that the element has a "void" label, while a "+" means selected, and a "-" stands for deselected. The specific notation used is irrelevant for now, but will be discussed in another example (see Figure 10). For "Serial No" the first rule applies, as both concept and attribute are selected. In (b) it can be seen that the connection between these two also receives the "selected" label. The connection between "Brand

Name" and "Car" receives a "deselected" label, as one of the elements (i.e. "Brand Name") is deselected. Finally, nothing is changed to the connection between "Car" and "Model Name" (i.e. it keeps its "void" label), as the two first rules do not apply here. Alternatively, it can be said that the third rule applies.

3.2 Optimization Criteria Related to Quality of an Ontology

There are many possible optimization schemes that seek to optimize one or more criteria that are related to the quality of the ontology. However at the outset it is necessary to make precise the notion of the quality of an ontology and the related criteria that might be used in any optimization scheme to achieve this quality. Generally one can note that quality of an ontology is multifaceted and the emphasis that one might give to a particular facet will be subjective.

We have identified the following facets related to the quality of the materialized ontology view. Namely:

(i) Consistency of the Ontology with the Requirements
(ii) Well Formedness of the Ontology
(iii) Semantic Completeness of the Ontology
(iv) Overall Simplicity of the Ontology

Each of these facets will separately be used as an optimization criterion and lead to an optimization scheme that optimizes with respect to this criterion and these are discussed in turn below, in the remainder of this section. The list presented is not a full list, as it is possible to construct new optimization schemes according to specific needs that might not be catered for through a combination of any of the existing optimization schemes.

The goal is to arrive at an ontology that is optimized for a certain application (or group of applications). To achieve this, first a number of optimization schemes are presented, i.e. the separate components for specifying what is considered an optimized ontology in a particular case. These components are then applied on the base ontology in the specified order. This means that they work as a chain, one scheme taking the output from a previous scheme as its input. Obviously these separate components have to be able to communicate with each other, and the labeling introduced in section 3.1 is used for this.

Because of the labeling (providing a standard language for communication) and the independence of the optimization schemes (as each optimization scheme can function on its own) it is easy to integrate new optimization schemes. The optimizations schemes presented here are intended to be used on the ontology standards as defined previously. Note that other standards (such as DAML-OIL, or OWL) can reuse many of the optimization schemes presented here, as well as future ones developed for various standards (although sometimes with small modifications). Next we discuss the optimization schemes with respect to the above four criteria in turn.

3.2.1 Requirement Consistency Optimization Scheme (RCOS)

This optimization scheme ensures that the requirements as expressed by a user or optimization scheme (through the applied labeling) are consistent, i.e. there are no contradicting statements in the labeling. If contradicting statements are present, there is no way for an algorithm – or human user for that matter – to determine what the intention is.

Here, a number of rules will be specified to ensure that a solution set is consistent in its requirements.

RC Rule 1

$$\text{Given an ontology } O=<C, A, attr, B, M_a, M_b> \text{ with a labeling } \sigma \text{ applied to } O \quad (28)$$

$$\forall b \in B:\ \sigma_C(\pi_1(b))=deselected \vee \sigma_C(\pi_2(b))= deselected \Rightarrow \sigma_B(b) \neq selected$$

A binary relationship provides additional meaning to how two concepts relate to each other. Having $\sigma_B(b)=$ *selected* indicates that this meaning has to be used in the 'target' ontology. However, if both concepts that are related to each other by *b* are deselected, this is saying that no information about these concepts should be present in the 'target' ontology. RC rule 1 sets out how these two statements are utilized in development of the optimized ontology.

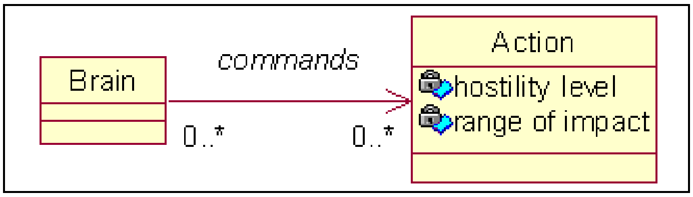


Fig. 7. UML Representation of an Ontology (Small Part)

A small example of the rule in practice is given through Figure 7. Applying two sets of labels – given in Table 2 – produces a valid and invalid combination according to this rule.

Table 2. Valid and Invalid Labeling for RCOS Rule 1

Valid Labeling (void labels not shown)	Invalid Labeling (void labels not shown)
$\sigma_C(Brain)=deselected$ $\sigma_B(commands)=deselected$	$\sigma_C(Brain)=deselected$ $\sigma_B(commands)=selected$

RC Rule 2

$$\text{Given an ontology } O=<C, A, attr, B, M_a, M_b> \text{ with a labeling } \sigma \text{ applied to } O \quad (29)$$

$$\forall t \in attr:\ \sigma_C(\pi_1(t))=deselected \vee \sigma_A(\pi_2(t))= deselected \Rightarrow \sigma_{attr}(t) \neq selected$$

This transformation rule is the equivalent to RC Rule 1, applied to the special relationship between concepts and their attributes, which is represented in the mapping *attr*.

RC Rule 3

$$\textit{Given an ontology } O=<C, A, attr, B, M_a, M_b> \textit{ with a labeling } \sigma \textit{ applied to } O \tag{30}$$

$$\forall t \in attr: \sigma_A(\pi_2(t)) = selected \Rightarrow \sigma_{attr}(t) \neq deselected$$

Here the more specific characteristic of the attribute-concept relationship is considered. No contradicting statements between the attribute and the connection are allowed, and this transformation rule together with the previous one ensures this.

Before the next rule can be given, first the notion of a path needs to be introduced and defined. Paths are very important in the specification of ontology views, as they introduce new relationships that are semantically linked to the original relationships.

Definition 17

$$\begin{aligned}&\textit{Given an ontology } O=<C, A, attr, B, M_a, M_b>\\ &\textit{we define}\\ &\quad \textit{a path } p \textit{ of } O \Leftrightarrow p \equiv b_1, b_2, \ldots, b_n \in B^+, \textit{ with } n \in \mathbb{N}_0\\ &\textit{such that}\\ &\quad \pi_1(b_i)=\pi_2(b_{i-1}), \textit{ with } i \in [2, n]\end{aligned} \tag{31}$$

Definition 18

$$\begin{aligned}&\textit{Given an ontology } O\\ &\textit{We define}\\ &\quad Path(O) \equiv \{\textit{set of all possible paths in } O\}\end{aligned} \tag{32}$$

Notation 7

$$\textit{Given an ontology } O=<C, A, attr, B, M_a, M_b> \tag{33}$$

$$\textit{And a path } p \in Path(O), \textit{ with } p = b_1, b_2, \ldots, b_n \in B^+ \; (n \in \mathbb{N}_0)$$

we denote

$$\pi_1(p)=\pi_1(b_1)$$

$$\pi_2(p)=\pi_2(b_n)$$

Definition 19

$$\begin{aligned}&\textit{Given an ontology } O=<C, A, attr, B, M_a, M_b> \textit{ and}\\ &\textit{a labeling } \sigma \textit{ applied to } O\\ &\textit{we define}\\ &\quad \textit{a proper path } p \Leftrightarrow p \equiv b_1, b_2, \ldots, b_n \in B^+ \; (n \in \mathbb{N}_0)\\ &\textit{with}\\ &\quad p \in Path(O)\\ &\quad \sigma_c(\pi_2(p)) \neq \textit{'deselected'}\\ &\quad \sigma_B(b_j) \neq \textit{'deselected'}, \textit{ with } j \in [1,n]\end{aligned} \tag{34}$$

Definition 20

$$\begin{gathered}\textit{Given an ontology } O \\ \textit{We define} \\ PPath(O) \equiv \{\textit{set of all possible proper paths in } O\}\end{gathered} \tag{35}$$

The additional definition for a more restricted set of paths *PPath(O)* is given as these are the types of paths that are considered mostly for the concatenation and replacement cases, which will be discussed further on.

From the previous definitions it follows that the *PPath(O)* is a subset of *Path(O)* ($PPath(O) \subset Path(O)$).

To give a couple of examples of valid and invalid types of paths, a labeled ontology graph is presented in Figure 8.

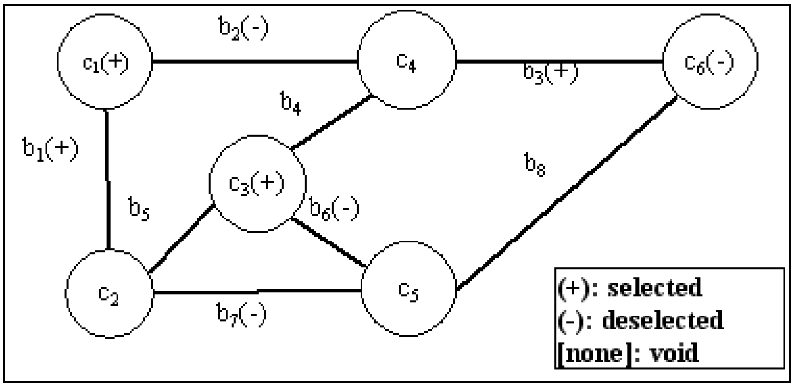


Fig. 8. Example Labeled Ontology Graph

Table 3 gives a number of paths and indicates why certain paths are invalid. Note that although not present in this table, it is possible for a valid proper path to connect a vertex (i.e. concept) that is deselected, as long as it is not the last vertex in the chain.

Table 3. Example Valid and Invalid Paths

Path	Validity	Description
$b_7b_6b_5b_1b_3b_8$	Invalid path	$\pi(b_1) \neq \pi(b_3)$ $\rightarrow$No concept that is connected by b_1 is connected by b_3.
$B_7b_6b_5b_1b_2b_4$	Valid path	
$B_7b_6b_5b_1b_2b_4$	Invalid Proper Path	$\sigma_B(b_7)=\sigma_B(b_6)=\sigma_B(b_2)$=deselected $\rightarrow$Some relationships in the path are labeled 'deselected'.
$b_5b_4b_3$	Invalid Proper Path	$\pi_2(b_3)=c_6$=deselected $\rightarrow$target concept is deselected.
$b_3b_4b_5b_1$	Valid Proper Path	

Now that the definitions for a path and a proper path have been given, the final RCOS rule can be presented. This rule specifies that if an attribute is selected, but the concept is deselected, i.e. there will be a need for distributing the attribute, there should be a proper path to a concept that is not deselected (so the attribute can potentially be placed there).

RC Rule 4

Given an ontology $O=<C, A, attr, B, M_a, M_b>$ *with a labeling* σ *applied to* O (36)

$$\forall t \in attr, \exists p \in PPath(O):$$

$$\sigma_A(\pi_2(t))=\text{'selected'} \wedge \sigma_C(\pi_1(t))=\text{'deselected'}$$

$$\Rightarrow \sigma_C(\pi_2(p)) \neq \text{'deselected'} \wedge \pi_1(p)= \pi_1(t)$$

An example of the combination described in the antecedent is presented in Figure 9, where the concept "Writer" is deselected, while one of its attributes ("genre") is selected. To adhere to the rule, a proper path p needs to exist that has its final concept not deselected, and that starts from the concept "Writer".

In this particular example there are two possible proper paths that can be used, and they are given below. Which one of these paths will eventually be utilized (or may be

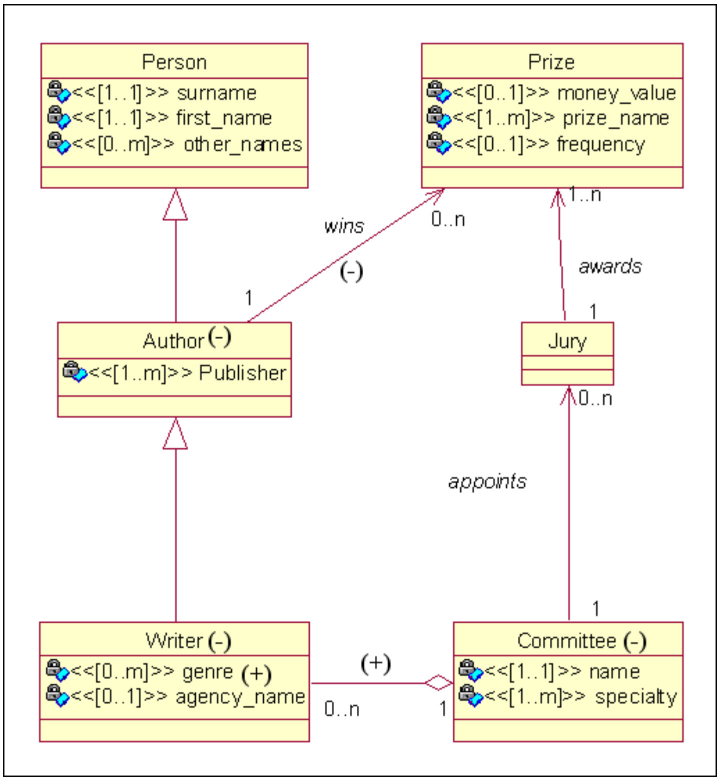


Fig. 9. Example Labeled UML Representation for RCOS Rule 4

even another possible solution might arise) is not important at this stage, merely that is possible to find at least one solution to the problem.

The first proper path that can be identified starts with the inheritance relationship between "Writer" and "Author". Because "Author" is deselected, this is not a valid end result for a path, but if the next inheritance relationship is included in the path (from "Author" to "Person"), a valid proper path is obtained, as none of the relationships is deselected, nor is the target concept ("Person"). Another possibility is the path that starts at "Writer", then goes to "Committee", and then crosses to "Jury". There are more possibilities (e.g. by extending the two paths already obtained), but as it is known for certain now that there is at least one proper path, the rule is satisfied.

3.2.2 Well Formedness Optimization Scheme (WFOS)

It might be clear what the intentions are of a certain labeling, but there possibly are statements that inevitably lead to a solution set that is not a valid ontology. The WFOS contains the proper rules to prevent this from happening.

Effectively, all the separate criteria as given in section 2 are interpreted here. It would suffice to say that per definition the set consisting of the selected elements of O (according to labeling σ) has to be a valid ontology, but here we want to establish the rules that will not only ensure this, but also are written in relation to the original ontology, i.e. the complete labeling.

A first, basic requisite for an ontology was $C\neq\emptyset$. In the specific case for the solution set we get:

$$\begin{aligned}
&\sigma_C(C)\neq\emptyset \\
\Rightarrow\quad &\exists c\in C\text{: } \sigma_C(c)=\text{'selected'} \vee \sigma_C(c)=\text{'void'} && \text{(Definition } \sigma_C(C)) \\
\Rightarrow\quad &\exists c\in C\text{: } \sigma_C(c)\neq\text{'deselected'} && \text{(Definition } S)
\end{aligned}$$

Intuitively, as there are only three possibilities for the mapping, of which the solution set contains two, and following from the criterion that there has to be one concept in the solution set for it to be a valid, well formed ontology, it is clear that at least one concept has to have a mapping that is other than 'deselected'.

This result is left as such and put into a first rule:

WF Rule 1

Given an ontology $O=<C, A, attr, B, M_a, M_b>$ with a labeling σ applied to O (37)

$$\exists c\in C\text{: } \sigma_C(c)\neq\text{'deselected'}$$

Another requirement that is present in the definition for an ontology is $\forall a\in A, \exists c\in C\text{: } a\in attr(c)$, stating that every attribute must belong to a concept. Starting from the adapted version a rule that could be postulated is:

$$\begin{aligned}
&\forall a\in\sigma_A(A), \exists c\in\sigma_C(C)\text{: } a\in\sigma_{attr}(\,attr)(c) \\
\Rightarrow\quad &\forall a\in\sigma_A(A), \exists c\in\sigma_C(C), \exists t\in\sigma_{attr}(attr)\text{: } t(c)=a && \text{(def. Ontology)} \\
\Rightarrow\quad &\forall a\in A, \text{ with } a=t(c) \text{ and } t\in attr, c\in C\text{:} && \text{(def } \sigma(O)) \\
&\neg(\sigma_A(a)=deselected) \Rightarrow \neg(\sigma_C(C)=deselected) \wedge \neg(\sigma_{attr}(t)=deselected) \\
\Rightarrow\quad &\sigma_C(c)=deselected \vee \sigma_{attr}(t)=deselected \Rightarrow \sigma_A(a)=deselected \\
&(*)
\end{aligned}$$

However this rule is not in satisfactory form as an additional requirement must also be considered. This additional requirement will then be combined with the above unsatisfactory rule to develop a basis for the additional rules.

The other requirement for the ontology that will be used next is $Attr{:}C \rightarrow 2^{A}$. Adapted to the well formedness of a solutions set, this gives;

$\sigma_{attr}(attr){:}\ \sigma_C(C) \rightarrow 2\sigma^{C(A)}$

$\Rightarrow$ *Given an attribute mapping* $t=(c, a)$ *with* $c \in C$, $a \in A$

$\neg(\sigma_{attr}(t)=deselected) \Rightarrow \neg(\sigma_C(c)=deselected) \wedge \neg(\sigma_A(a)=deselected)$

$\Rightarrow$ $\sigma_C(c)=deselected \vee \sigma_A(a)=deselected \Rightarrow \sigma_{attr}(t)=deselected$

(**)

Combining the above two results we obtain:

(*) $\wedge$ (**)

$\Rightarrow$ 1) $\sigma_C(c)=deselected \Rightarrow \sigma_{attr}(t)=deselected$

2) $\sigma_A(a)=deselected \Rightarrow \sigma_{attr}(t)=deselected$

3) $\sigma_C(c)=deselected \Rightarrow \sigma_A(a)=deselected$

4) $\sigma_{attr}(t)=deselected \Rightarrow \sigma_A(a)=deselected$

1) $\wedge$ 3) $\Rightarrow$ $\sigma_C(c)=deselected \Rightarrow \sigma_A(a)=deselected \wedge \sigma_{attr}(t)=deselected$

2) $\wedge$ 4) $\Rightarrow$ $\sigma_A(a)=deselected \Leftrightarrow \sigma_{attr}(t)=deselected$

These last statements are taken as the rules for the well formedness optimization scheme, thus:

WF Rule 2

Given an ontology O=<C, A, attr, B, Ma, Mb> with a labeling σ applied to O (38)

$$\forall t \in attr{:}\ \sigma_C(\pi_1(t))=deselected \Rightarrow \sigma_A(\pi_2(t))=deselected \wedge \sigma_{attr}(t)=deselected$$

WF Rule 3

Given an ontology $O=<C, A, attr, B, M_a, M_b>$ *with a labeling σ applied to O* (39)

$$\forall t \in attr{:}\ \sigma_A(\pi_2(t))='deselected' \Leftrightarrow \sigma_{attr}(t)='deselected'$$

In Figure 10 an example of these two rules is shown. As the attribute mapping is not explicitly catered for in UML (it is not a link, but a container, i.e. attributes are contained inside a class), a different notation (semantic net notation [Feng, Chang et al. 2002]) for the example was used. The concept "Person" is deselected, and so is according to the antecedent of WF Rule 2. The attribute "surname" is selected (and its connection to "Person" has no indication, so it's the default label, i.e. "void"), and that is invalid for WF Rule 2. The attribute "first_name" and its attribute mapping are both deselected, giving a valid case for WF Rule 2. As "Document" has the default – void – label, the antecedent is not met, so every case including that concept is valid according to WF Rule 2.

However, WF Rule 3 says both attributes and attribute mappings for the concept "Document" result in invalid combinations, as both the attribute as its attribute

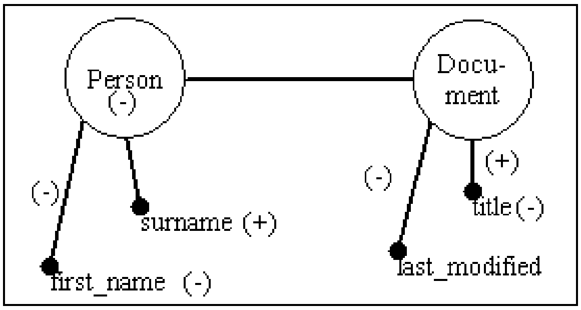


Fig. 10. WFOS Valid and Invalid Cases in Semantic Net Notation [Feng, Chang et al. 2002]

mapping should be deselected (if one of them is deselected). Looking at the attributes for concept "Person", both of them are valid. The attribute "surname" does not trigger WF Rule 3, and "first_name" does, but is a valid combination.

The next requirement for an ontology as given in the definition is $B \subseteq C \times C$. Put in the proper context this can be stated as:

$$\sigma_B(B) \subseteq \sigma_C(C) \times \sigma_C(C)$$
$$\Rightarrow \quad \text{given a binary relation } b=(c_1, c_2) \text{ with } c_1, c_2 \in C$$
$$\sigma_B(b) \neq \text{'deselected'} \Rightarrow \sigma_C(c_1) \neq \text{'deselected'} \wedge \sigma_C(c_2) \neq \text{'deselected'}$$
$$\Rightarrow \quad \sigma_C(c_1) = \text{'deselected'} \vee \sigma_C(c_2) = \text{'deselected'} \Rightarrow \sigma_B(b) = \text{'deselected'}$$
$$\Rightarrow \quad (1)\ \sigma_C(c_1) = \text{'deselected'} \Rightarrow \sigma_B(b) = \text{'deselected'}$$
$$(2)\ \sigma_C(c_2) = \text{'deselected'} \Rightarrow \sigma_B(b) = \text{'deselected'}$$

(1) and (2) are now used to define the next rule which is as follows:

WF Rule 4

Given an ontology O=<C, A, attr, B, M_a, M_b> with a labeling σ applied to O

$$1)\ \forall b \in B: \sigma_C(\pi_1(b)) = \text{'deselected'} \Rightarrow \sigma_B(b) = \text{'deselected'} \tag{40}$$

$$2)\ \forall b \in B: \sigma_C(\pi_2(b)) = \text{deselected'} \Rightarrow \sigma_B(b) = \text{'deselected'}$$

Additional rules could be provided here for the n-ary relationship, but through transformations of the design n-ary relationships can be written as binary relationships, so the same rules would apply to these relationships.

To be complete, the ontology graph requirement for an ontology is repeated here, although it is not developed into another rule, but rather kept in its original form.

WF Rule 5

$$\text{Given an ontology } O \text{ with a labeling } \sigma \text{ applied to } O \tag{41}$$

$$\exists \text{ an ontology graph } G\sigma_{(O)} \text{ for } \sigma(O)$$

An example of an invalid ontology is given in Figure 11. Although it may not be clear immediately (consider the difficulty if not impossibility of visually recognizing

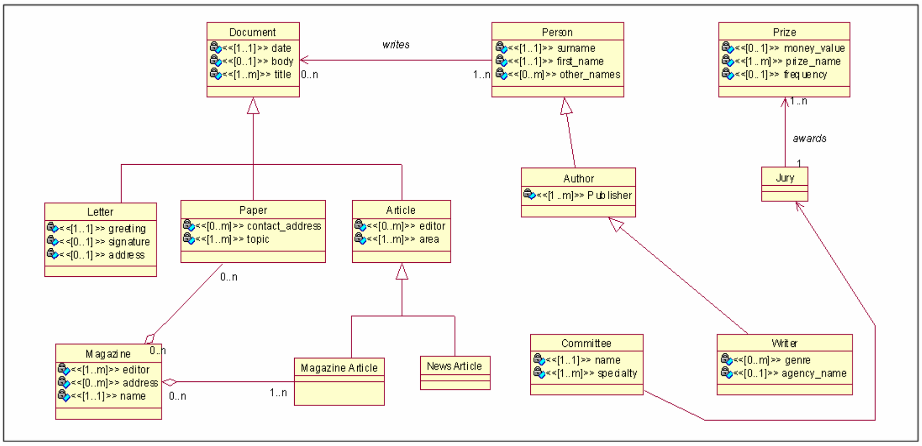


Fig. 11. UML Representation of Invalid Ontology

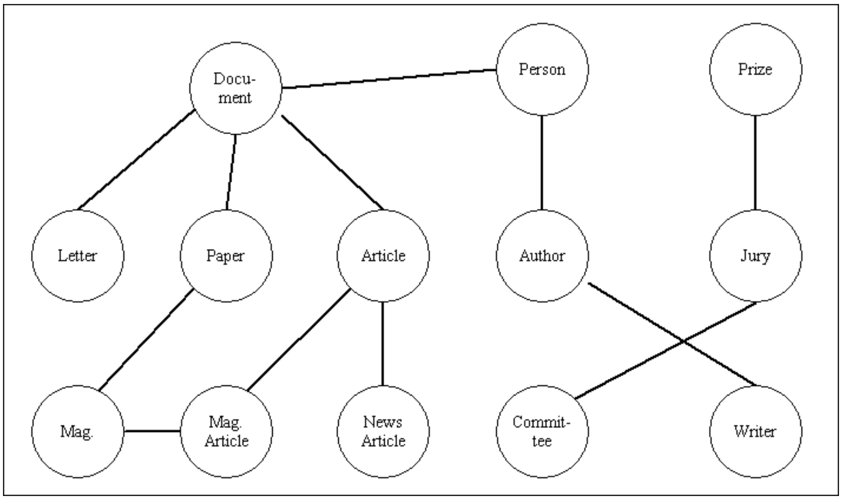


Fig. 12. Ontology Graph Representation of the Ontology (Invalidity Unclear)

a similar situation in real – complex – ontologies), even not when looking at the corresponding ontology graph (in Figure 12), this ontology is invalid as its ontology graph contains an island (or proper component in graph theory terms). This becomes clear when the vertices from the ontology graph are rearranged (no information changed, just the visual coordinates).

The fifth WFOS Rule clearly requires a valid ontology graph, and a valid ontology graph according to its definition cannot contain any islands. An alternative representation of the ontology graph is shown in Figure 13. They boxed gray area is a clear separate island, and so the resulting ontology graph (that is obtained from the ontology) is invalid, and so the ontology is invalid as well (not well formed, i.e. not adhering to WF Rule 5).

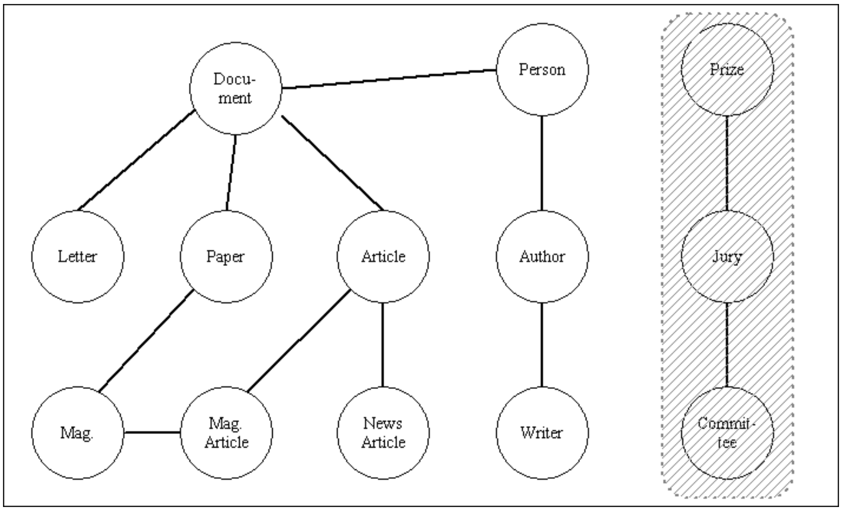


Fig. 13. Ontology Graph Representation of the Ontology (Invalidity Clear)

This optimization scheme, together with the previous one, ensures the minimal quality that is necessary for any materialized ontology view (that it is in fact a valid ontology, and that the criteria for the optimization are clear). Any meaningful extraction process should always include these two optimization schemes.

3.2.3 Semantic Completeness Optimization Scheme (SCOS)

The previous schemes were obvious, and are the minimal schemes that will be applied (except maybe in some rare cases), but the following optimization schemes, starting with the SCOS are not always wanted, or even possible concurrently. Not all the schemes are given here, and it is possible to introduce new schemes for new needs, however, the schemes given demonstrate clearly the manner in which quality – or optimization – is introduced in the extraction process.

The SCOS considers the completeness of the concepts, i.e. if one concept is defined in terms of another concept, the latter cannot be omitted without losing some semantic meaning of the former concept. Including these defining elements is not always required, but in those cases this optimization scheme should not be selected, as the ordered list of optimization schemes can be freely composed by an ontology engineer.

Firstly, some additional definitions need to be made to specify what is meant by 'defining elements'. In the definition for an ontology the set of binary relationships was further split up in different types of relationships. For some of these sets the direction of the relationship is important, so now a more detailed function π is introduced to serve the purpose of distinguishing between the linked concepts.

Additional information on the notation of some general concepts such as sub-super classes, whole-part aggregation, etc. is given next to facilitate further discussion.

Notation 8

Given an ontology O=<C, A, attr, B, M_a, M_b> with a labeling σ applied to O

$$\begin{gathered} B = B_i \cup B_{agg} \cup B_r,\ \forall c_1, c_2 \in C \\ \textit{We denote} \\ c_1 \textit{ is a subconcept of } c_2 \\ \Leftrightarrow \in b \in B_i\text{: } \pi_1(b)=c_1 \wedge \pi_2(b)=c_2 \end{gathered} \tag{42}$$

Notation 9

Given an ontology O=<C, A, attr, B, M_a, M_b> with a labeling σ applied to O

$$\begin{gathered} B = B_i \cup B_{agg} \cup B_r,\ \forall c_1, c_2 \in C \\ \textit{We denote} \\ c_1 \textit{ is a superconcept of } c_2 \\ \Leftrightarrow \in b \in B_i\text{: } \pi_1(b)=c_2 \wedge \pi_2(b)=c_1 \end{gathered} \tag{43}$$

In the example shown in Figure 14 (a) "Tree" is the sub-concept ($\pi_1(b_1)$), while "Plant" is the super-concept ($\pi_2(b_1)$).

Notation 10

Given an ontology O=<C, A, attr, B, M_a, M_b> with a labeling σ applied to O

$$\begin{gathered} B = B_i \cup B_{agg} \cup B_r,\ \forall c_1, c_2 \in C \\ \textit{We denote} \\ c_1 \textit{ is a part of } c_2 \\ \Leftrightarrow \in b \in B_{agg}\text{: } \pi_1(b)=c_1 \wedge \pi_2(b)=c_2 \end{gathered} \tag{44}$$

Notation 11

Given an ontology O=<C, A, attr, B, M_a, M_b> with a labeling σ applied to O

$$\begin{gathered} B = B_i \cup B_{agg} \cup B_r,\ \forall c_1, c_2 \in C \\ \textit{We denote} \\ c_1 \textit{ contains } c_2 \\ \Leftrightarrow \in b \in B_{agg}\text{: } \pi_1(b)=c_2 \wedge \pi_2(b)=c_1 \end{gathered} \tag{45}$$

This aggregation relationship is shown in UML notation in Figure 14(b). The concept "Computer" is the whole-concept ($\pi_2(b_2)$), while "CPU" is the part-concept ($\pi_1(b_2)$)

Notation 12

Given an ontology O=<C, A, attr, B, M_a, M_b> with a labeling σ applied to O

$$\begin{gathered} B = B_i \cup B_{agg} \cup B_r,\ \forall c_1 \in C,\ \forall a_1 \in A \\ \textit{We denote} \\ a_1 \textit{ is a defining attribute for } c_1 \\ \Leftrightarrow \exists t \in attr\text{: } t=(c_1, a_1) \wedge \exists m \in card : \neg(m_{min}(t)=0) \end{gathered} \tag{46}$$

Figure 14(c) shows an example of a defining attribute. A "Person" has to have at least (here exactly) one "surname".

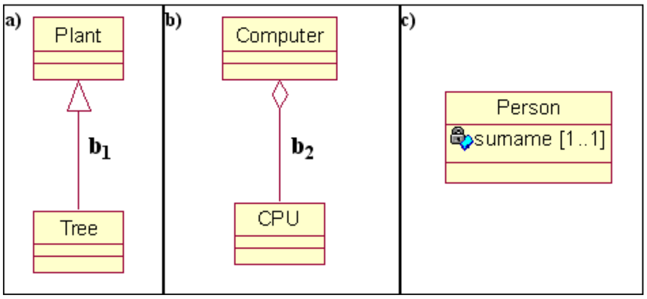


Fig. 14. Example Semantic Essential Elements

Because ontologies allow for such a diverse way of defining new elements (e.g. a new class that is the union of two existing concepts), a simplified approach is taken here. Although we will show only a limited number of ways for developing rules, extending this basic set is straight forward. As mentioned before, optimization schemes can also be constructed for various standards.

The first defining way considered here is the super-concept of another concept.

SC Rule 1

Given an ontology O=<C, A, attr, B, M_a, M_b> with a labeling σ applied to O

$$\forall b \in B: \sigma_C(\pi_1(b))=selected \Rightarrow \sigma_C(\pi_2(b))=selected \wedge \sigma_B(b)=selected \tag{47}$$

This rules states that whenever a concept is retained for a view, the superconcepts of it, and the actual inheritance relationship stating this are required as well to be semantically complete.

The following rule does the same for the part relationship, i.e. when a concept is required (mapped to 'selected') then part concepts of this concepts are retained as well.

SC Rule 2

Given an ontology O=<C, A, attr, B, M_a, M_b> with a labeling σ applied to O, and a binary aggregation relationship $b \in B$ (48)

$$\sigma_C(\pi_2(b))=selected \Rightarrow \sigma_C(\pi_1(b))=selected \wedge \sigma_B(b)=selected$$

Finally, also a rule is given here for the 'not null' attributes of a concept, as they provide information that according to the ontology specification should always be there (an empty field is not possible). Although 'defining' might be too strong a word for these attributes, they are nonetheless required for there to be a semantically complete ontology view.

SC Rule 3

Given an ontology O=<C, A, attr, B, M_a, M_b> with a labeling σ applied to O, an attribute mapping t, and a cardinality m for t (49)

$$\sigma_C(\pi_1(t))=selected \wedge m_{min}\neq 0 \Rightarrow \sigma_A(\pi_2(t))=selected \wedge \sigma_{attr}(t)=selected$$

3.2.4 Total Simplicity Optimization Scheme (TSOS)

The TSOS is an applied and modified version of Kruskal's Algorithm [Kruskal 1956] for minimal spanning trees. Applying this optimization scheme will result in the smallest possible solution that is still a valid ontology (starting from a certain solution set). This optimization scheme is included as an example of how certain requirements (smallest possible solution) are translated in an optimization scheme. Furthermore, it clearly shows as well that not all optimization schemes are desirable in every situation. Sometimes the most versatile solution is sought, not the smallest, and thus the TSOS would not be applied. The choice of what is considered important or high quality is represented by the selection of optimization schemes, and the order in which they are applied.

To get to a solution for the TSOS, a new labeling is introduced. This labeling only applies to the element previously labeled 'void'. When the methods of achieving this optimization scheme are discussed, the reasoning behind the additional labeling will be clarified. First, definitions for different sets are given to aid in the readability of the following rules and definitions.

Definition 21

Given an ontology $O=<C, A, attr, B, M_a, M_b>$ with a labeling σ applied to O

$$\begin{aligned} B' &\equiv \{b \in B \mid \sigma_B(b)=void\} \\ C' &\equiv \{c \in C \mid \sigma_C(c)=void\} \\ A' &\equiv \{a \in A \mid \sigma_A(a)=void\} \end{aligned} \tag{50}$$

Definition 22

Given an ontology $O=<C, A, attr, B, M_a, M_b>$ with a labeling σ applied to O

$$\begin{aligned} B'' &\equiv \{b \in B \mid \sigma_B(b)=selected\} \\ C'' &\equiv \{c \in C \mid \sigma_C(c)=selected\} \\ A'' &\equiv \{a \in A \mid \sigma_A(a)=selected\} \end{aligned} \tag{51}$$

For the new labeling that is needed, the labeling σ could be extended, but as not all targets are necessary, and for clarity in notation, a new labeling δ is introduced here.

Definition 23

$$T \equiv \{accepted, rejected\} \tag{52}$$

The set of possible labels contains merely two elements, but these are on top of the δ labeling.

Definition 24

$$\begin{aligned} \delta_{C'} &: C' \rightarrow T \\ \delta_{B'} &: B' \rightarrow T \end{aligned} \tag{53}$$

Definition 25

$$\begin{gathered} B_a' \equiv \{b \in B' \mid \delta_{B'}(b) = accepted\} \\ B_r' \equiv \{b \in B' \mid \delta_{B'}(b) = rejected\} \\ C_a' \equiv \{c \in C' \mid \delta_{C'}(c) = accepted\} \\ C_r' \equiv \{c \in C' \mid \delta_{C'}(c) = rejected\} \end{gathered} \tag{54}$$

Now that the differently labeled elements have been grouped in sets, the rules that follow become more readable. These rules represent what is understood by a total simplified solution. As mentioned before, Kruskal's Algorithm for minimal spanning trees is taken as a starting point, but as no weights are present, labels are used to replace them. Some of these labels – 'selected' and 'deselected' – have to be treated as fixed by the optimization scheme used, and so the best solution is sought by labeling the 'void' elements so that all of the following rules are adhered to.

TS Rule 1

$$\begin{gathered} \text{Given an ontology } O=<C, A, attr, B, M_a, M_b> \text{ with a labeling } \delta \text{ applied to } O, \\ <C_a' \cup C'', B_a' \cup B''> \text{ is a valid ontology graph} \end{gathered} \tag{55}$$

This rule can be considered a rule that gives the lower bound, as it limits the number of elements that can be rejected. The elements accepted by the algorithm, together with the earlier labeled 'selected' elements still have to form a valid ontology graph.

An example of a seemingly valid ontology, but which is in reality invalid (because it has an invalid ontology graph), was shown in Figure 11, Figure 12 and Figure 13.

The next rule, on the other hand, determines the upper bound, stating that elements that have been labeled 'accepted' should always be necessary to have a valid ontology graph. If not, they should have been labeled 'rejected' instead, as the resulting valid ontology graph (after they are rejected) represents a smaller – i.e. more simplified – solution.

TS Rule 2

$$\begin{gathered} \text{Given an ontology } O=<C, A, attr, B, M_a, M_b> \text{ with a labeling } \delta \text{ applied to } O, \\ \text{And an ontology graph } G_O=<V,E> \text{ for } O \\ \forall V' \subseteq C_a, E' \subseteq B_a, O_1 \in \vartheta: <V \setminus V', E \setminus E'> \text{ is not a valid ontology graph for } O_1 \\ \text{with} \\ V' \neq \varnothing \end{gathered} \tag{56}$$

At first, this rule seems a bit complicated, but intuitively, it states that the obtained two tuple ($<V\setminus V', E\setminus E'>$) is not a valid ontology graph for any ontology (O_1). In other words, the two tuple does not conform to all the requirements set out in the definition for an ontology graph. It is straight forward to convert elements of V\V' to concepts, and elements of E\E' to relationships. This lack of validity could be due to; i) the two tuple is not a valid graph, or ii) the two tuple is a valid graph, but contains islands (proper components).

The second case speaks for itself, but the first case needs clarification. From the elements we have in the sets an invalid graph can be obtained if at least one edge connects at least one vertex that is not an element of V\V' ($\exists e \in E\setminus E' : e=(v_1,v_2) \Rightarrow v_1 \vee v_2 \notin V\setminus V'$).

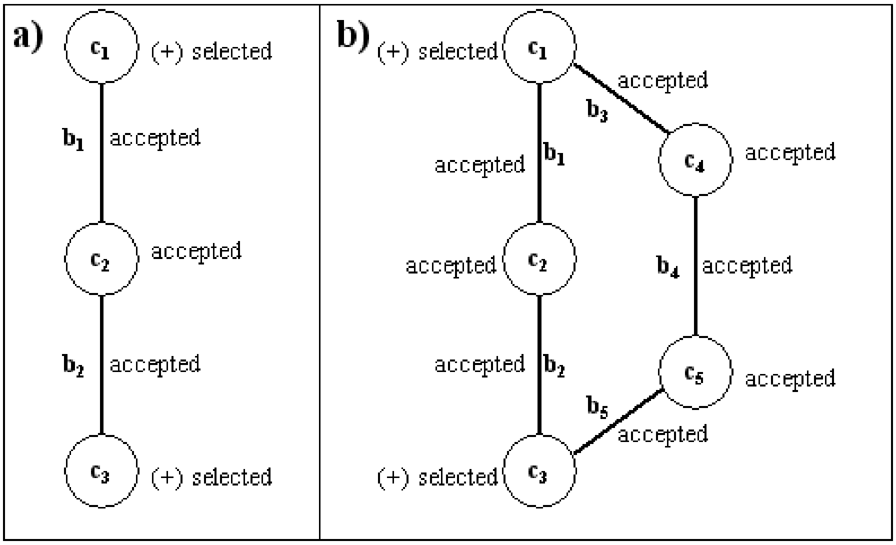


Fig. 15. Two Valid Graphs with Accepted and Selected Elements Exclusively

An example of applying this rule is demonstrated with the aid of Figure 15, showing two graphs with the same selected elements, but graph b) introduces a number of additional accepted elements. It is clear that graph a) is simpler than graph b), and so graph b) is not optimal and therefore not a solution (assuming they both aim to provide a solution for the same base ontology in the same extraction process). The previous rule can be applied, and so any combination of accepted elements can be sought, then taken away from the graph, and if the result is a possible valid ontology graph, the starting graph does not adhere to the rule. For graph a), there are only three accepted (so 6 possible combinations, i.e. $\{(b_1), (b_2), (c_2), (b_1c_2),(b_1b_2),(b_2c_2)\}$, as the order is irrelevant). For graph b) however, a lot more combinations are possible, and an example can be found of a combination of accepted elements that – when they are left out – leaves a valid ontology graph. An example is the set of elements $\{b_1,b_2,c_2\}$, or $\{b_3,b_4,b_5,c_4,c_5\}$. Both these sets can be left out and the result would still be a valid ontology graph (for a certain ontology).

The last rule is a very short and strict rule, ensuring the simplest solution. It eliminates the possibility of having an attribute still labeled as a 'void'. The 'void' label would indicate that it is still undecided what will happen to the attribute, but as the simplest solution is sought, all the attributes that were undecided should be deselected now, as they are not necessary, and thus not part of the simple solution that is the goal.

TS Rule 3

Given an ontology $O=<C, A, attr, B, M_a, M_b>$ with a labeling σ applied to O,

$$\forall a \in A : \neg(\sigma_A(a) = void \tag{57}$$

Note that potentially more than one solution is possible for a given solution set. Looking back at the non optimal ontology graph presented in Figure 15 (graph b), two different sets were given, both of which could be left out to result in a still valid and optimal result. The two outcomes of these operations are both possible solution sets. Through a weighting system a more efficient connection can be preferred (see Appendix A for a more detailed discussion of possible weights of relationships), but then it is still possible that multiple potential results have the same total sum of the weights, and thus can be considered equally optimized solutions. A practical example of this selection of best paths can be found in [Wouters, Dillon et al. (to appear December 2003)].

TSOS is different from previous optimization schemes in that it leaves a completely decided solution set, i.e. one without 'void' elements. It is only useful to put such an optimization scheme once in a sequence of optimization schemes, and only towards the end, as it would not leave an undecided element for the next optimization scheme to work with. A practical example of such a sequence of optimization schemes (called priority list) can be found in [Wouters, Dillon et al. (to appear December 2003)].

3.3 Dynamic Rules Necessary for Algorithmic Development

The previous sections set out the key definitions and the rules which define what has to be achieved in deriving optimized materialized ontology views from a base ontology. To facilitate automation of the process of extraction of materialized ontology views we need to develop algorithms which carry out the implementation of the rules. In other words, they set the 'how' rather than the 'what' of materialized ontology view extraction. As a prelude to actually working out the algorithms, we will specify a set of rules that provide the basis for the algorithms. The rules given here have a more dynamic character than the ones given in the previous section. The dynamic transformation of what replacements, modifications and even extensions can be made to achieve the static requirements is the main focus of this section.

An example is the attribute riddance rule given before. It just states that no more 'void' labels are allowed for the attributes. How we should go about achieving this is demonstrated by algorithms, such as in this case; "replace all the 'void' labels with 'deselected' labels for the TSOS".

First of all, three general rules are given in their most generic form. Further on, these rules will be utilized in algorithms.

3.3.1 Mapping Modification

This generic definition shows how the mapping in general can be modified, and thus has no impact on the actual ontology. The ontology remains exactly as it was before, i.e. there is no transformation from an ontology O to an ontology O', but only from a mapping σ to a mapping σ'. As in the case of the other rules, this rule is also defined as a transformational function with certain input parameters.

Mapping Modification Definition

Given

an ontology $O=<C, A, attr, B, M_a, M_b>$ with a labeling σ applied to O,

$X \in \{C, A, attr, B\}$, (58)

$s_1, s_2 \in S$,

$x \in X$, and

$\sigma_X(x)=s_1$

we define

$\xi_X\sigma(x, s_2) = \sigma' = (\sigma \backslash \{\sigma_X(x)\}) \cup \sigma_X'(x)$

with

$\sigma_X'(x) = s_2$

Intuitively, this definition specifies that an existing mapping can be modified, resulting in a new mapping. The modification only needs an element and a label, and the resulting mapping is the same as the original, except that the label for the specified element has been changed to the new label.

3.3.2 Attribute Distribution

There already has been a mention of the necessity for redistributing attributes to obtain an optimized result (section 3.2.1). The definition of how this transformation is done is given here.

Attribute Distribution Definition

Given

an ontology $O = <C, A, attr, B, M_a, M_b>$,

$t \in attr$,

$b \in B$

with

$\pi_1(t) = \pi_1(b)^{(1)} \vee \pi_1(t) = \pi_2(b)^{(2)}$

we define

$\xi_A^O(b, t) \equiv O' \equiv <C', A', attr', B', M_a', M_b'>$

with

$A' = A$

$C' = C$

$B' = B$ (59)

$M_b' = M_b$

(1) $\underline{\pi_1(t) = \pi_1(b)}$

$attr' = (attr \backslash \{t\}) \cup \{t'\}$

with $t' = (\pi_2(b), \pi_2(t))$

$M_a' = (M_a \backslash \{m(\pi_2(t))\}) \cup \{m'(\pi_2(t))\}$

with $m'(\pi_2(t)) = (m_{min}(\pi_2(t)) * m_{min1}(b), m_{max}(\pi_2(t)) * m_{max1}(b))$

(2) $\underline{\pi_1(t) = \pi 2(b)}$

$attr' = (attr \backslash \{t\}) \cup \{t'\}$

with $t' = (\pi_1(b), \pi_2(t))$

$M_a' = (M_a \backslash \{m(\pi_2(t))\}) \cup \{m'(\pi_2(t))\}$

with $m'(\pi_2(t)) = (m_{min}(\pi_2(t)) * m_{min2}(b), m_{max}(\pi_2(t)) * m_{max2}(b))$

This definition specifies how attributes can 'travel' across a binary relationship, and what the consequences are for the cardinality of the concerned attribute.

3.3.3 Relationship Concatenation

Often it is required to drop a certain concept from a solution, but still the semantic link it provides to relationships that bridge two other concepts might be relevant. The notion of a path, which was introduced previously, is essential to the concatenation rule.

Relationship Concatenation Definition

Given

$$\text{an ontology } O=<C, A, attr, B, M_a, M_b>,$$

$$\text{a path } p= b_1, b_2, \ldots, b_n \in B^+ \ (n\in \mathbb{N}_0)$$

we define

$$\xi_B^O(p)\equiv O'\equiv<C', A', attr', B', M_a', M_b'>$$

with

$$\begin{aligned}
&A'=\{a\in A | a=\pi_2(t) \wedge \pi_1(t)\neq\pi_1(b_i), \text{ with } t\in attr, i\in[2,n]\}\\
&attr'=\{t\in attr | \pi_1(t)\neq\pi_1(b_i), \text{ with } i\in[2,n]\}\\
&C'= \{c\in C | c\neq\pi_1(b_i), \text{ with } i\in[2,n]\}\\
&B'=\{b\in B | b\neq b_i, \text{ with } i\in[2,n]\}\\
&M_a'=\{m\in M_a | m(a)=(n_1, n_2), \text{ with } a\in A', n_1, n_2\in card\}\\
&M_b'=\{m\in M_b | m(b)=(n_1, n_2, n_3, n_4), \text{ with } b\in B', n_1, n_2, n_3, n_4\in card\}\cup\{m(b')\}\\
&\quad \text{with}\\
&\qquad b'=(\pi_1(p), \pi_2(p)) \wedge\\
&\qquad m(b')=(m_{min1}(b_1)*m_{min1}(b_2)*\ldots*m_{min1}(b_n),\\
&\qquad\qquad m_{max1}(b_1)*m_{max1}(b_2)*\ldots*m_{max1}(b_n),\\
&\qquad\qquad m_{min2}(b_1)*m_{min2}(b_2)*\ldots*m_{min2}(b_n),\\
&\qquad\qquad m_{max2}(b_1)*m_{max2}(b_2)*\ldots*m_{max2}(b_n))
\end{aligned} \tag{60}$$

Having a valid path as an input, the resulting ontology replaces the linking concepts and relationships by a new one, calculates the cardinality for the new relationship, and discards all the attributes of the replaced concepts. The multiplication table for the cardinality set has to be specified, otherwise results cannot be calculated (see Appendix A).

4 Development of the Algorithms

The previous sections have shown what the requirements for different optimization schemes are, and what transformations we are allowed to use to modify a labeled ontology so that it complies with the requirements. This section links those two together, and for the relevant optimization schemes algorithms are given that show step by step how an application – with minimal human interaction required – can arrive at an adequate result for an optimization scheme.

4.1 Requirement Consistency Optimization Scheme

The rules set out for the RCOS ensure there is no contradiction between statements in the input user requirements. In other words, this optimization scheme checks for combinations of labels that lead to a situation that is dubious or ambiguous for the system, and that cannot be resolved without more clarification. This clarification needs to come from the user. Note that this clarification process can be automated as well, by having a rule system that can be applied in cases of inconsistency.

An example of such an automation can be that in case of inconsistency, elements labeled as ‘selected’ always take preference over elements labeled as ‘deselected’. In case of RC rule 1 we could have a relationship *b* with a ‘selected’ labeling, but with $\pi_1(b)$ labeled as ‘deselected’. In other words, the end result should definitely have the relationship, but it should not contain one of the concepts that is being related to another by the relationship. Clearly there is no solution that satisfies both of these, so the requirements are inconsistent. Without any further information an automated system would not be able to resolve this inconsistency, but if we take into account the preference rule that was added, the inconsistency can be easily resolved.

Note that this optimization scheme always is the first to be applied, and all the other optimization schemes rely on the fact that there are no inconsistencies in the requirements anymore.

The first three rules of the RCOS require algorithms that are very similar to algorithms for rules of other optimization schemes. For this reason they are not given here, but by slightly modifying other algorithms the necessary algorithms are readily obtainable.

The fourth rule needs a more complex and unique algorithm to enforce it. It uses the notion of a proper path discussed earlier.

4.1.1 RC Rule 4

```
1. Loop through all $c \in C$
    1.1. if ($\sigma_C$(c)== ‘deselected’)
        1.1.1. loop through all attributes $a \in A$ that have an attributemapping to c
            1.1.1.1. if ($\sigma_A$(a)== ‘selected’)
                1.1.1.1.1. if FIND_PROPER_PATH(c) returned false
                           // no proper path was found
                    1.1.1.1.1.1. NO SOLUTIONS
                    1.1.1.1.1.2. exit algorithm
```

Although seemingly very brief here, this is only because the sub algorithm FIND_PROPER_PATH was used. The particular algorithm for finding such a proper path is not given here, but an indication of how it operates can be found in the example given in Figure 9.

4.2 Well Formedness Optimization Scheme Algorithm

The algorithm shows how an ontology and mapping are checked and modified to comply with the WFOS. Every rule is sequentially visited to get to the final result, and the rules are identified in the algorithm as well, for improved understandability of the algorithm. Some input ontologies and mappings have no solutions that adhere to all requirements given by the WFOS. In the algorithm these cases are indicated as we come across them. To resolve these cases, a similar solution to the one discussed in the previous section can be used sometimes, but are not considered here.

In the algorithm it will indicated if certain cases are not considered because they are not present thanks to the prerequisite of the RCOS – remember that RCOS is a prerequisite to most optimization schemes, including WFOS.

4.2.1 WF Rule 1

```
1. if ($\sigma_C == \emptyset$)
    1.1. NO SOLUTIONS (all the concepts are deselected, so the end result will be empty)
```

4.2.2 WF Rule 2

```
1. While ($\exists t \in attr$: $\sigma_C(\pi_1(t))$ == 'deselected' $\wedge$ ($\sigma_A(\pi_2(t)) \neq$ 'deselected' $\vee$ $\sigma_{attr}(t) \neq$ 'deselected') )
    1.1. if there is no proper path p from $\pi_1(t)$ to another concept $c_1$ that is not 'deselected' ($\sigma_C(c_1) \neq$ 'deselected')
        1.1.1. if ($\sigma_A(a)$ == 'void')          // never 'selected' because RC4
            1.1.1.1. $\sigma \leftarrow \xi_A^{\sigma}$(a, 'deselected')
    1.2. else (there is one (p) or more ({$p_1, p_2, \ldots, p_n$}) proper paths)
        1.2.1. if multiple paths
            1.2.1.1. p $\leftarrow$ BEST_PATH({$p_1, p_2, \ldots, p_n$})
        1.2.2. loop i from 1 to n          // with p=$b_1 b_2 \ldots b_n$
            1.2.2.1. O $\leftarrow \xi_A^{O}(b_i, t)$
            1.2.2.2. t $\leftarrow$ t'          // t' is constructed in Attribute Distribution Rule
```

In step 1.2.1.1 another algorithm BEST_PATH is called. This algorithm determines the 'best' path that is available. What is considered a good or strong path is an extensive topic, and the algorithm used here is given in Appendix A. Note that this is only one possible solution, and it is not the intention of the authors to claim this is the best or only solution. In reality it will depend on what preferences are emphasized, and a good solution for that particular case can be found, but no general 'best' solution. The algorithm for BEST_PATH supplied in the appendices suited our needs, and thus is provided merely as an example of one possibility amongst many.

Note that the algorithms that are given here are not meant to be the most efficient ones. The algorithms here serve the purpose of demonstrating how they resolve encountered problems, not how they can (or should) be implemented most efficiently.

4.2.3 WF Rule 3

```
2. Loop through all $t \in attr$
    2.1. if ($\sigma_A(\pi_2(t))$ == 'deselected' $\wedge$ $\sigma_{attr}(t)$ == 'void')
                                   // never $\sigma_{attr}$(t)='selected' because RC2
        2.1.1. $\sigma \leftarrow \xi_{attr}^{\sigma}$(t, 'deselected')
    2.2. else
        2.2.1. if ($\sigma_{attr}(t)$ == 'deselected' $\wedge$ $\sigma_A(\pi_2(t))$ == 'void')
                                   // never $\sigma_A(\pi_2(t))$='selected' because RC3
            2.2.1.1. $\sigma \leftarrow \xi_A^{\sigma}(\pi_2(t)$, 'deselected')
```

As previously mentioned, not all possible cases are considered here, as the RCOS is a requirement, rendering some cases from not occuring here. The appropriate rules have been stated.

4.2.4 WF Rule 4

```
1. loop through all $b \in B$
   1.1. if $(((\sigma_C(\pi_1(b)) ==$ 'deselected' $\vee\ \sigma_C(\pi_2(b)) ==$ 'deselected') $\wedge\ \sigma_A(\pi_2(t)) ==$ 'void')
        1.1.1. $\sigma \leftarrow \xi_B^{\sigma}(b,$ 'deselected')
```

4.2.5 WF Rule 5

The main requirements for this rule (WF rule 5) are always correct, as they are just an assignment of certain elements to a new set. However, the third, "no islands in the graph" requirement is important here.

```
1. if (ISLANDS_EXIST($\sigma$(O)) == true)
   1.1. NO SOLUTIONS (a 'deselect' labeling prevents an ontology graph from
        being constructed)
```

The ISLANDS_EXIST algorithm uses Kruskal's algorithm to try to get to a minimal spanning tree, and if such a minimal spanning tree cannot be accomplished, there is an island. The algorithm attempts to find a minimal spanning tree, and such a tree has an interesting characteristic that is used here; if successful, the number of vertices is always equal to the number of edges plus one. As the resulting graph is acyclic (a minimal spanning tree cannot be cyclic), we know that there are never going to be more edges than the amount of vertices minus one. The third possibility is that there are less edges than the number of vertices minus one, indicating there must be at least two islands (if there's only one island, it is the entire ontology, logically). A simple example of these three cases is shown in Figure 16.

For a more thorough explanation of these characteristics of graphs, please refer to [Biggs, Lloyd et al. 1976], as these characteristics are taken here in a matter-of-fact manner, without explanations or proofs.

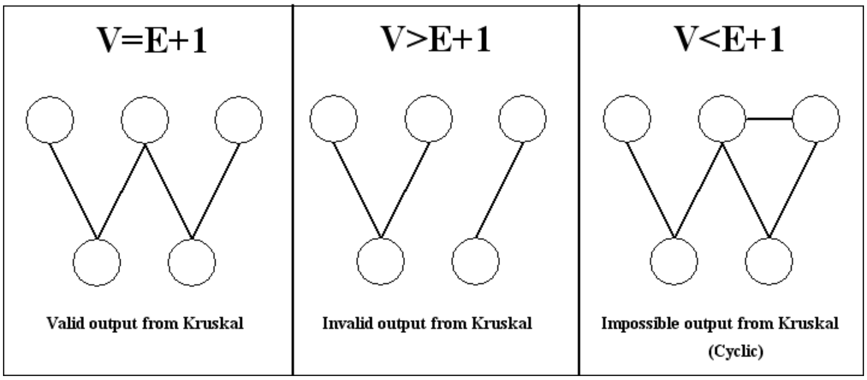

V=E+1
V>E+1
V<E+1
Valid output from Kruskal
Invalid output from Kruskal
Impossible output from Kruskal (Cyclic)

Fig. 16. The Three Cases for an Ontology Graph

4.3 Semantic Completeness Optimization Scheme Algorithm

As in the previous algorithm every rule taken from the requirements is visited and receives its own algorithm. In all the cases it is possible that through a 'deselected' label of a certain element no semantic completeness can be obtained. A mention of this particular case is made in the algorithms. It does not lead to an exit from the loop, but an indication that the ideal solution was not possible. In other words, if the ideal solution is not possible, the algorithm will inform the system of this, but it will still continue, so that the final result is as close to the ideal solution as possible (given the input labeling). The same type of preference rules as the one shown in section 4.1 can be used here, but to focus on that here would distract from the main focus of this paper.

4.3.1 SC Rule 1

```
1. $B_{temp} \leftarrow B_i$    // $B_i$ is set of all binary inheritance relationships
2. while ($B_{temp} \neq \emptyset$)
    2.1. $b_{temp} \leftarrow B_{temp}[0]$
    2.2. $b_{sub} \leftarrow$ FIND_SUB($B_{temp}$, $\pi_1(b_{temp})$)
    2.3. while ($b_{sub} \neq \emptyset$)
        2.3.1. $b_{temp} \leftarrow b_{sub}$
        2.3.2. $b_{sub} \leftarrow$ FIND_SUB($B_{temp}$, $\pi_1(b_{temp})$)
    2.4. if ($\sigma_C(\pi_1(b_{temp}))$ == 'selected')
        2.4.1. if ($\sigma_B(b_{temp})$ == 'deselected' $\vee$ $\sigma_C(\pi_2(b_{temp}))$ == 'deselected')
            2.4.1.1. NO SC POSSIBLE
        2.4.2. else
            2.4.2.1. $\sigma \leftarrow \xi_B^{\sigma}(b_{temp}$, 'selected')
            2.4.2.2. $\sigma \leftarrow \xi_C^{\sigma}(\pi_2(b_{temp})$, 'selected')
    2.5. $B_{temp} \leftarrow B_{temp} \setminus \{b_{temp}\}$
```

Note that the additional algorithm FIND_SUB is a simple loop that goes through the first input parameter and tries to find a relationship that has the second input parameter as the superconcept of another concept. This (inheritance) relationship is then returned, or null if none was found. This algorithm is omitted from this article.

4.3.2 SC Rule 2

```
1. $B_{temp} \leftarrow B_{agg}$    // $B_{agg}$ is the set of all the binary aggregation relationships
2. while ($B_{temp} \neq \emptyset$)
    2.1. $b_{temp} \leftarrow B_{temp}[0]$
    2.2. $b_{whole} \leftarrow$ FIND_WHOLE($B_{temp}$, $\pi_2(b_{temp})$)
    2.3. while ($b_{whole} \neq \emptyset$)
        2.3.1. $b_{temp} \leftarrow b_{whole}$
        2.3.2. $b_{whole} \leftarrow$ FIND_WHOLE($B_{temp}$, $\pi_2(b_{temp})$)
    2.4. if ($\sigma_C(\pi_2(b_{temp}))$ == 'selected')
        2.4.1. if ($\sigma_B(b_{temp})$ == 'deselected' $\vee$ $\sigma_C(\pi_1(b_{temp}))$ == 'deselected')
            2.4.1.1. NO SC POSSIBLE
        2.4.2. else
            2.4.2.1. $\sigma \leftarrow \xi_B^{\sigma}(b_{temp}$, 'selected')
            2.4.2.2. $\sigma \leftarrow \xi_C^{\sigma}(\pi_1(b_{temp})$, 'selected')
    2.5. $B_{temp} \leftarrow B_{temp} \setminus \{b_{temp}\}$
```

Very similar to FIND_SUB, the FIND_WHOLE algorithm finds the 'whole' concept of a 'part' input parameter, and because of the straight forward nature of the algorithm, is omitted here.

4.3.3 SC Rule 3

```
1. Loop through concepts $c \in C$ that are selected ($\sigma_C(c)$ == 'selected')
    1.1. Loop through all $t \in attr$ with $\pi_1(t)=c$
        1.1.1. if ($m_{min}(\pi_2(t)) \neq 0$)
            1.1.1.1. if ($\sigma_A(\pi_2(t))$ == 'deselected' $\vee$ $\sigma_{attr}(t)$ == 'deselected')
                1.1.1.1.1. NO SC POSSIBLE
            1.1.1.2. else
                1.1.1.2.1. $\sigma \leftarrow \xi_A^{\sigma}(\pi_2(t)$, 'selected')
                1.1.1.2.2. $\sigma \leftarrow \xi_{attr}^{\sigma}(t$, 'selected')
```

4.4 Total Simplicity Optimization Scheme Algorithm

The Total Simplicity Optimization Scheme is different from the other optimization schemes in a number of ways, as already mentioned in section 3.2.4. Another difference with the other optimization schemes that are presented in this paper, is that the rules to adhere to are very clear as to what is optimal, but do not let themselves translate into algorithms so straight forwardly as other optimization scheme rules. Therefore, another approach is taken here, rather than providing transparent algorithms per rule.

The algorithms were first based on Kruskal's algorithm, but a more custom solution to the problem was needed, as Kruskal does not incorporate fixed elements, such as our selected and deselected elements. Inspired by the firs attempt, ontology graphs are still taken as the model to work on, instead of the actual ontology. Note that an abstraction between the ontologies and ontology graphs is made, i.e. first the ontology graph is constructed from the ontology, but from there on changes to the ontology graph do not automatically incur similar changes to the actual ontology. For instance, deleting a vertex from the ontology graph will not result in the corresponding concept being deleted from the ontology, but instead indicates a modification of the labeling of the concept to 'deselected'. To clarify these differences between operations on the ontology graph and changes that occur to the actual ontology, in the algorithm the different implications will both be described (impact on the ontology always in brackets).

To simplify the algorithm, certain combinations of elements and their labels, are explained separately, instead of in the algorithm. Also, redistribution of the attributes is not covered here, but more information on this can be found in Appendix A. After the algorithm has finished with the other elements (relationships and concepts), this redistribution of attributes takes place, according to the reallocation via the best path, as used previously.

TS algorithm

```
1. Construct ontology graph
2. Leave out all deselected elements
3. While combination1 is present
    3.1. Make the edge selected (same for corresponding relationship in the ontology)
```

```
4. While combination2 is present
    4.1. Make void vertex selected (same for corresponding concept in the ontology)
5. While combination3 is present
    5.1. Replace combination3 with a new vertex with a selected label (no change to
         ontology
    5.2. If there is another void edge between the vertices
        5.2.1. Remove edge (corresponding relationship becomes deselected)
    5.3. Else    // edge is selected
        5.3.1. Remove edge (no change to the ontology)
6. While there are void vertices left, choose one (called v)
    6.1. if connectivity ==1
        6.1.1. if connecting edge has void label
            6.1.1.1. Remove edge and vertex (deselect corresponding relationship
                     and concept)
        6.1.2. else    // this means edge is selected (deselected ones are already
               removed)
            6.1.2.1. Make vertex selected (same for corresponding concept)
            6.1.2.2. if the edge given combination3
                6.1.2.2.1. Replace combination3 with a new selected vertex (no
                           changes to the ontology)
    6.2. Else    // connectivity > 1
        6.2.1. n ← connectivity
        6.2.2. call edges $e_1, \ldots, e_n$
        6.2.3. call connected vertices $v_1, \ldots, v_n$
        6.2.4. delete v and $e_1, \ldots, e_n$ (make corresponding concept and relation-
               ships deselected in the ontology)
        6.2.5. create $C_2^n$ new void edges, i.e. one between every possible combi-
               nation of $v_1, \ldots, v_n$ (create corresponding relationships in ontology,
               with a void label)
        6.2.6. if there are multiple edges between any vertices of $v_1, \ldots, v_n$
            6.2.6.1. delete edge(s) with higher weight(s) (the corresponding rela-
                     tionships becomes deselected in the ontology)
7. If any changes to the ontology graph were made, repeat step 3 -6
```

First, the 4 specific combinations that are used throughout the algorithm will be given here.

- ***combination1***: A void edge connecting two selected vertices
- ***combination2***: A selected edge with at least one void vertex
- ***combination3***: A selected edge connecting two selected vertices

Besides the four combinations, the algorithm uses the connectivity of vertices, which is defined as the number of edges that are connected to a vertex. As a valid ontology does not contain any islands, this means that the vertices of the corresponding ontology graph always have at least one edge, and in the algorithm care is taken that when the last edge of a vertex is deleted, the vertex is removed as well (and appropriate actions are taken for their impact on the ontology). A final comment about the algorithm concerns step 6.2.6.1 where the edges with a higher weight are removed (only the minimum weight is retained). What the weights are, and how they are calculated for newly constructed edges, relies on the cardinalities of the relationships. More information about this can be found in Appendix A. As discussed previously, there are many ways in which the weight of a relationship can be calculated, leading to very different results. Here only one method is shown, merely as an example of a possibility.

5 Conclusion

Transforming ontologies is becoming an important factor in the success of the semantic web. Ontologies tend to grow larger, and this results in an extremely tedious process – maybe to the point of being impossible – if an ontology needs to be modified (e.g. new version, sub-ontology, distribution, materialized ontology view). The formalisms presented in this paper provide a way in which automated extraction of materialized ontology views becomes possible, thus preventing problems associated with the manual extraction from occurring. After some general definitions, and introducing some new concepts, optimization schemes were presented as a means of arriving at high-quality or optimized results. As the notion of quality is multifaceted, various optimization schemes were introduced, and the theory of how these different optimization schemes can be used as building blocks by ontology engineers to arrive at a solution that they consider optimized, was covered. How the rules of optimization schemes could be implemented was demonstrated by first providing some dynamic rules that could be used by a system, in the form of transformation functions, and then utilizing these transformation functions in algorithms that ensure the rules of every optimization scheme are adhered to.

At the moment, only a limited number of optimization schemes has been developed, and in future work this will be addressed, firstly by the development of more optimization schemes, but also by setting a standard of communication and working of such an optimization scheme, so anyone can use these guidelines to develop optimization schemes that can be dynamically loaded into an Extraction Framework. This will lead to a plugin architecture where optimization schemes for various standards can be used in a single framework, and where contributions to the library of optimization schemes can be made by everyone.

References

Berners-Lee, T., Al: Reference Description of the DAML+OIL Ontology Markup Language (2001)

Biggs, N.L., Lloyd, E.K., et al.: Graph Theory 1736-1936. Clarendon Press, Oxford (1976)

Chen, P.P.: The Entity-Relationship Model: Toward a Unified View of Data. ACM Transaction on Database Systems 1(1), 9–36 (1976)

Colomb, R.M., Weber, R.: Completeness and Quality of an Ontology for an Information System. In: Proceedings of International Conference on Formal Ontology In Information Systems, Trento, Italy (1998)

Date, C.J.: An Introduction to Database Systems. Addison Wesley, Reading (2000)

Feng, L., Chang, E., et al.: A Semantic Network Based Design Methodology for XML Documents. ACM Transactions on Information Systems 20(3) (2002)

Fensel, D.: The Semantic Web, Tutorial Notes. In: 9th IFIP 2.6 Working Conference on Database Semantics (2001)

Fensel, D., Decker, S., et al.: Ontobroker: Or How to Enable Intelligent Access to the WWW. In: Proceedings 11th Knowledge Acquisition for Knowledge-Based Systems Workshop, Banff, Canada (1998)

Fensel, D., Horrocks, I., et al.: OIL in a Nutshell. In: Proceedings 12th International Conference on Knowledge Engineering and Knowledge Management Methods, Juan-Les-Pins, France (2000)

Genesereth, M.R.: Knowledge Interchange Format. In: Proceedings of the Second International Conference on Principles of Knowledge Representation and Reasoning. Morgan Kaufmann Publishers, San Francisco (1991)

Genesereth, M.R., Fikes, R.: Knowledge Interchange Format, version 3.0, reference manual. Computer Science Department, Stanford University, Stanford (1992)

Gruber, T.R.: Ontolingua: A Mechanism to Support Portable Ontologies. Knowledge Systems Laboratory, Stanford University, Stanford (1992)

Gruber, T.R.: Toward principles for the design of ontologies used for knowledge sharing. In: Guarino, N., Poli, R. (eds.) Formal Ontology in Conceptual Analysis and Knowledge Representation. Kluwer Academic Publishers, Dordrecht (1993)

Guarino, N., Welty, C.: Evaluating Ontological Decisions with OntoClean. Communications of the ACM 45(2), 61–65 (2002)

Hahn, U., Schnattinger, K.: Towards Text Knowledge Engineering. In: Proceedings of the 15th National Conference on Artificial Intelligence, Madison, Wisconsin (1998)

Halpin, T.: Conceptual Schema and Relational Database Design. Prentice Hall, Englewood Cliffs (1995)

Heflin, J., Hendler, J.: Dynamic Ontologies on the Web. In: Proceedings of American Association for Artificial Intelligence Conference, Menlo Park, California (2000)

Heflin, J., Hendler, J., et al.: SHOE: A Knowledge Representation Language for Internet Applications, Dept. of Computer Science, University of Maryland (1999)

Holsapple, C.W., Joshi, K.D.: A Collaborative Approach to Ontology Design. Communications of the ACM 45(2), 42–47 (2002)

Hovy, E.H.: Combining and Standardizing Large-Scale, Practical Ontologies for Machine Translation and Other Uses. In: Proceedings of the First International Conference on Language Resources and Evaluation, Granada, Spain (1998)

Kaplan, A.N.: Towards a Consistent Logical Framework for Ontological Analysis. In: Proceedings of the International Conference on Formal Ontology in Information Systems (2001)

Kim, H.: Predicting How Ontologies for the Semantic Web will Evolve. Communications of the ACM 45(2), 48–54 (2002)

Klein, M., Fensel, D.: Ontology versioning for the Semantic Web. In: Proceedings of the International Semantic Web Working Symposium, California, USA (2001)

Klein, M., Fensel, D., et al.: Ontology versioning and Change Detection on the Web. In: Proceedings of the 13th International Conference on Knowledge Engineering and Knowledge Management, Sigüenza, Spain. Springer, Heidelberg (2002)

Kruskal, J.B.J.: On the shortest spanning subtree of a graph and the traveling salesman problem. Proc. American Mathematics Society (7), 48–50 (1956)

Lenat, D.B.: Cyc: A Large-Scale Investment in Knowledge Infrastructure. Communications of the ACM 38(11) (1995)

McGuinness, D.L., Fikes, R., et al.: An environment for merging and testing large ontologies. In: Proceedings of the Seventh International Conference on Principles of Knowledge Representation and Reasoning. Morgan Kaufmann, San Francisco (2000)

Muslea, I.: Extraction Patterns for Information Extraction Tasks: A Survey. In: AAAI 1999 Workshop on Machine Learning for Information Extraction (1999)

Noy, N.F., Hafner, C.D.: The State of the Art in Ontology Design. AI-Magazine (Fall), 53–74 (1997)

Noy, N.F., Sintek, M., et al.: Creating Semantic Web Contents with Protégé-2000. IEEE Intelligent Systems 16(2), 60–71 (2001)

Rumbaugh, J., Jacobson, I., et al.: Unified Modeling Language Reference Manual. Addison-Wesley, Reading (1999)

Spyns, P., Meersman, R., et al.: Data modelling versus Ontology engineering. SIGMOD (special issue), 14–19 (2002)

Van Harmelen, F., Fensel, D.: Practical Knowledge Representation for the Web. In: Proceedings of International Joint Conferences on Artificial Intelligence (1999)

Von Staudt, G.K.C.: Geometrie der Lage. Nurnberg (1847)

W3C. Extensible Markup Language (XML) 1.0. W3C Recommendation (1999)

W3C. Feature Synopsis for OWL Lite and OWL. W3C Working Draft (2002a)

W3C. OWL Web Ontology Language 1.0 Abstract Syntax. W3C Working Draft (2002b)

W3C. OWL Web Ontology Language 1.0 Reference. W3C Working Draft (2002c)

Wouters, C., Dillon, T., et al.: A Practical Walkthrough of the Ontology Derivation Rules. In: Hameurlain, A., Cicchetti, R., Traunmüller, R. (eds.) DEXA 2002, vol. 2453. Springer, Heidelberg (2002a)

Wouters, C., Dillon, T., et al.: Transformational Processes for Sub-Ontologies Extraction (submitted for publication, 2002b)

Wouters, C., Dillon, T., et al.: A Practical Approach to the Derivation of Materialized Ontology View. In: Taniar, D., Rahayu, W. (eds.) Web Information Systems. Idea Group Publishing (to appear, December 2003)

Appendix A

A.1 Multiplication Table

The cardinality set used throughout this article is very limited (only consisting of three elements). This set can easily be extended, but in order for the optimization schemes to be able to use a new set, it needs to define a multiplication table. This is done in a lookup matrix, so that the solution for every possible multiplication can be found. The multiplication table for the cardinality set used here is given in Table 4.

Table 4. Multiplication Table for Cardinality Set

*	0	1	n
0	0	0	0
1	0	1	n
n	0	n	n

A.2 Best Path Algorithm

The Best Path Algorithm starts with a concept, and looks for the best possible path. In this context, ‘best’ means the lowest multiplication result of all the cardinalities (using the multiplication table). It is possible that more than one paths are returned as 'best' paths (i.e. multiple paths with the lowest weight). In this case, branches for each possibility are created, and treated as independent extraction processes. At the end of the extraction process, they are brought back together (and compared). In other words, it is possible that an extraction process produces several results, all equivalents in terms of quality.

```
Best_Path(concept $c_1$)
-   weight ← infinity
-   $P_{remaining}$ ← all relationships connected to $c_1$
-   While there are paths left in $P_{remaining}$
    o   Take first path in list (call it $p_i=b_{i0},\ldots,b_{ij}$)
    o   If $\sigma(b_{ij}) \neq$ deselected
        ▪   If $\pi_2(p_i) \neq$ deselected          // a solution found
            •   If $P_{sol} = \emptyset$
                o   Weight ← calc_weight($p_i$)
                o   $P_{sol}$ ← { $p_i$ }
            •   Else
                o   If (weight = calc_weight($p_i$))
                    ▪   $P_{sol}$ ← $P_{sol} \cup$ { $p_i$ }
                o   If (weight > calc_weight($p_i$))
                    ▪   $P_{sol}$ ← { $p_i$ }
                    ▪   Weight ← calc_weight($p_i$)
            •   Loop through $P_{remaining}$ (call it $p_n$)
                o   If (calc_weight($p_n$) > weight)
                    ▪   $P_{remaining}$ ← $P_{remaining}$ \ { $p_n$ }
        ▪   Else        // $\pi_2(p_i)$=deselected
            •   $R_{temp}$ ← all relationships connected to $\pi_2(p_i)$
            •   $R_{temp}$ ← $R_{temp}$ \ { $p_i$ }
            •   Loop through $R_{temp}$ (call it bx)
                o   $P_{remaining}$ ← $P_{remaining} \cup$ { $b_{i0},\ldots,b_{ij},b_x$ }
    o   $P_{remaining}$ ← $P_{remaining}$ \ { $p_i$ }
-   Return $P_{sol}$
```

As discussed previously, this is merely an example of an algorithm that was used throughout this article. It is not the intention of the authors to claim this is the most appropriate algorithm to use, it just worked well under conditions our testing was carried out.

A.3 Redistribution of Attributes

Attributes that are labeled 'selected', but belong to a deselected concept, need to be put somewhere else in the solution, in other words, a redistribution of attributes is required. This is done by looking for a proper path to a selected concept (which should exist, as this is one of the rules for RCOS), and if multiple alternatives exist, a similar algorithm to the previous one can be used to determine the 'best' option.

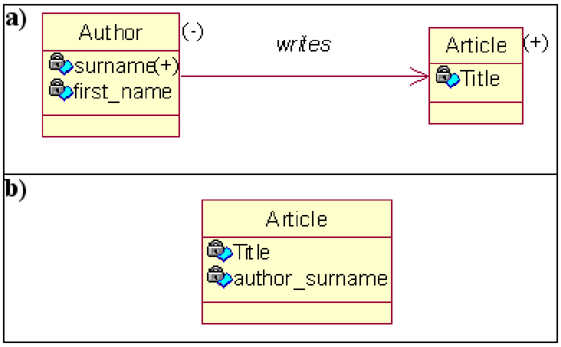


Fig. 17. Original (a) and Extracted (b) Ontology with Redistributed Attribute

Figure 17 shows an original ontology (a) where a deselected concept ("Author") has a selected attribute ("surname"). The second ontology (b) shows how the selected attribute was redistributed, and now belongs to the "Article" concept. Note that the name here has been changed to make the move of the attribute more meaningful.