Transformation from OWL Description to Resource Space Model*

Hai Zhuge, Peng Shi, Yunpeng Xing, and Chao He

China Knowledge Grid Research Group,
Key Laboratory of Intelligent Information Processing, Institute of Computing Technology,
Chinese Academy of Sciences, Beijing, 100080, China
{zhuge, pengshi, ypxing, hc}@kg.ict.ac.cn

Abstract. Semantics shows diversity in real world, document world, mental abstraction world and machine world. Transformation between semantics pursues the uniformity in the diversity. The Resource Space Model (RSM) is a semantic data model for organizing resources based on the classification semantics that human often use in understanding the real world. The design of a resource space relies on knowledge about domain and the RSM. Automatically creating resource space can relieve such reliance in RSM applications. This paper proposes an approach to automatically transform Web Ontology Language description into resource space. The normal forms of the generated resource space are investigated to ensure its normalization characteristic. The Dunhuang culture resource space is used to illustrate the approach.

1 Introduction

The machine-understandable semantics is commonly regarded as the key to the Semantic Web [3]. The W3C (www.w3.org) recommended the Web Ontology Language (OWL) in 2004 to support advanced Semantic Web applications by facilitating publication and sharing of ontology (www.w3.org/2004/OWL/).

The Resource Space Model (RSM) is a semantic model for uniformly specifying and organizing resources [16, 17]. It maps various resources (information, knowledge and services) into a multi-dimensional semantic space — a semantic coordinate system that normalizes the classification semantics. Each point in the space represents the resources of the same semantic category. Normal form and integrity theories of the RSM ensure correct semantic representation and operations [19]. The RSM theory is developed in parallel with the relational database theory [20].

The design of resource space is based on domain knowledge, application requirement and knowledge of RSM. The design method was proposed to guide the process of developing an appropriate resource space [18]. However, it still relies on designers' knowledge about domain and RSM. To relieve such reliance is an important issue of the RSM methodology.

The development of domain ontology makes codified domain knowledge. It will be very useful if we can codify the knowledge about RSM into an approach for auto-

* Keynote at ASWC2006. This work was supported by the National Basic Research Program of China (973 project No.2003CB317000) and the National Science Foundation of China.

R. Mizoguchi, Z. Shi, and F. Giunchiglia (Eds.): ASWC 2006, LNCS 4185, pp. 4 – 23, 2006.

matically transforming domain ontology into resource space. The semantics in OWL description can be used to support the creation of resource space.

This paper proposes an approach to automatically transform an OWL description into a resource space to enhance the efficiency of RSM design and relieve the reliance on individual knowledge by converting individuals of OWL to resources of RSM and transforming the inheritance hierarchy relationships and properties of resources into axes of RSM.

Relevant work includes the transformation between OWL service and the Unified Modeling Language (UML) [7], the converting from OWL ontology to UML [6], the bidirectional mapping between Attempto Controlled English (ACE) and OWL [10], converting from OWL DLP statements to logic programs [12], and the method for converting the existing thesauri and related resources from native format to RDF(S) or OWL [1]. A method reflecting the taxonomic relationship of products and service categorization standards (PSCS) in an OWL Lite ontology was proposed [9].

Related work also concerns software engineering area. The structural software development can be regarded as a multiple step transformation from the semantic specification on domain business into the semantic specification on software. Semantic specification tools like the Entity-Relationship model help developers transform domain business into relational model [4, 13]. The transformation from the E-R model into the relational database was investigated [2, 5, 15].

2 The Synergy of the Semantics in Real World, Document World and Abstraction World

Real world semantics used by human is hard to be understood by machines. Modeling languages like UML are for specifying real world semantics in standardized symbol systems.

Semantics in the mental world can be intuitive or abstract. Abstract semantics takes the form of symbolized and geometrical principles and theories. Human often use classification method to recognize the real-world. The implementation of the classification-based RSM depends on the data structures in the machine world, while the display of a resource space can be in the geometrical form of the abstraction world.

Semantics in the machine world is hard for ordinary people to understand. The XML, RDF and OWL mediate the machine world and the document world at different semantic levels.

Different semantics overlap and interact with each other to establish and develop the interconnection semantics as shown in Fig. 1. The future interconnection environment needs the synergy of the diversity and uniformity of semantics in the real world, the document world and the mental abstraction world. Automatic transformation between semantics of different levels is an important issue. The transformation from an OWL description to the RSM generalizes the semantics in the machine world and the document world. Since RSM is based on classification semantics, the created resource space (called OWL-based resource space) does not keep all the semantics described in OWL file. Transformations from OWL into abstract SLN and from UML into OWL are also significant.

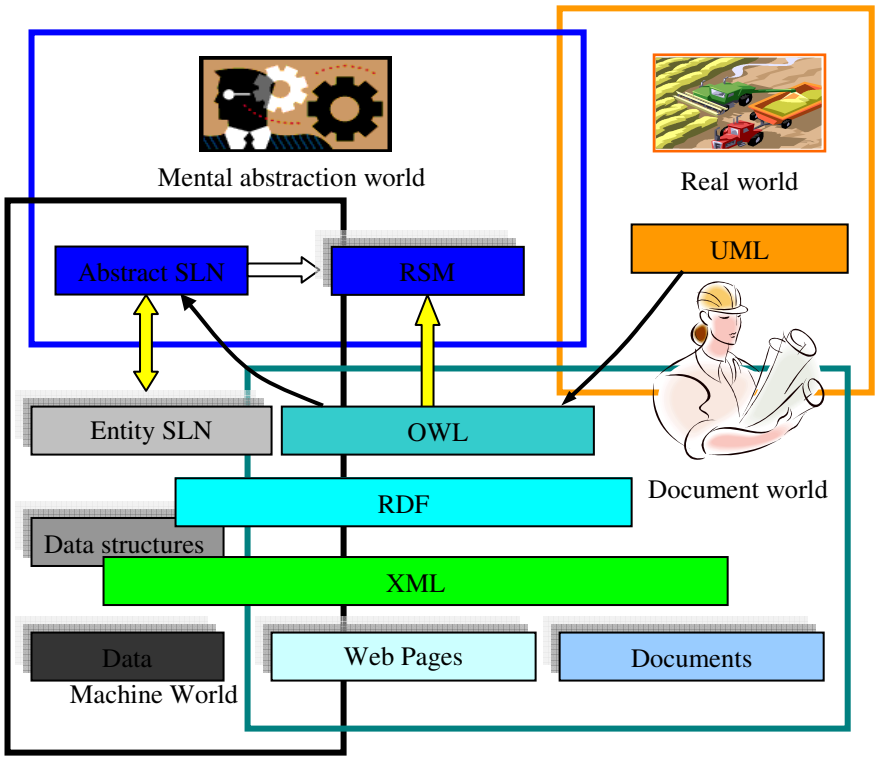


Fig. 1. The synergy of the semantics of four worlds in future interconnection environment

3 Basic Elements of OWL and an Example of RSM

Ontology facilitates the uniformity and sharing of domain knowledge by five types of basic modeling primitives: classes or concepts, relations, functions, axioms and instances [8, 14]. OWL provides three increasingly expressive sublanguages designed for specific users. OWL Lite is for the users who primarily need classification hierarchy and simple constraint features. OWL DL supports the maximum expressiveness without losing computational completeness and decidability of reasoning systems. OWL Full supports maximum expressiveness and the syntactic freedom of RDF without computational guarantees.

The following are basic elements of OWL:

(1) *Class* defines a class. An individual is an instance of a class.
(2) *rdfs*:*subClassOf* specifies the subclass relation.
(3) Properties are owned by classes or instances and divided into two types: *ObjectProperty* and *DatatypeProperty*. *ObjectProperty* specifies the relation between two instances, which belong to the same or different classes. *DatatypeProperty* indicates the relation between instance and RDF literals or XML Schema datatypes such as string, float or date.
(4) *rdfs*:*subPropertyOf* represents the inheritance of properties.

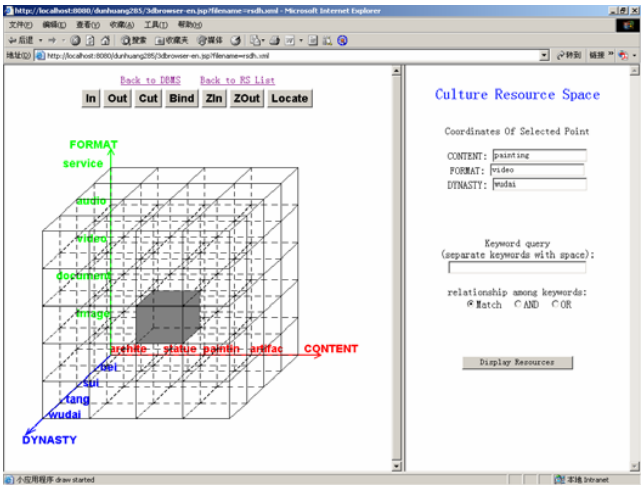


Fig. 2. The three-dimensional resource space browser

(5) *rdfs*:*domain* and *rdfs*:*range* restrict the anterior and posterior values of a property respectively. There are also some characteristics and restrictions, such as *TransitiveProperty*, *SymmetricProperty*, *allValuesFrom* and *Cardinality*, for describing property.

The following elements in OWL are used to improve the ability of describing the relations between classes, individuals and properties.

(1) *equivalentClass* and *equivalentProperty* represent the equivalence between classes and properties respectively.
(2) *sameAs* indicates that two individuals with different names are logically equal.
(3) *differentFrom* and *AllDifferent* explicitly distinguish one individual from others.
(4) *intersectionOf*, *unionOf* and *complementOf* are for set operation. They usually represent how a class is composed by other classes.
(5) *disjointWith* prevents a member of one class from being that of another class.

These elements help the mapping between different ontologies and describe more complex relationships between classes and individuals.

The most commonly used resource space is two- or three-dimensional, which can be displayed on screen and manipulated by users with ease. Fig.2 shows a three-dimensional resource space browser for Dunhuang culture exhibition. Resources can be located and manipulated by moving the black cube representing a point in the space. The black cube can be controlled by moving mouse and clicking the "In" and "Out" buttons.

4 Transformation from OWL Description into RSM

4.1 Process of Transformation

The main process of creating resource space from OWL file is shown in Fig.3. The input consists of the ancestor classes and the OWL file. The ancestor classes are the top-level classification of resources in an application.

The first step is to eliminate synonym in OWL file as the *equivalentClass*, *equivalentProperty* and *sameAs* in OWL may cause classification confusion when creating a resource space. A solution is to use one complex name to replace the synonyms.

Some individuals in OWL are transformed into resources in resource space. The inheritance hierarchy relationships and properties of resources in OWL are converted into axes of the OWL-based resource space.

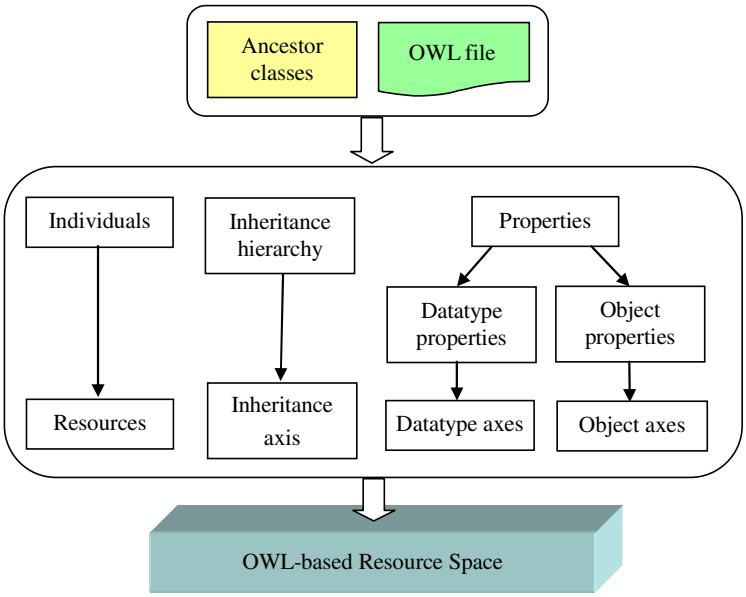


Fig. 3. The main process of creating resource space from OWL file

4.2 From Individuals to Resources

The individuals belonging to the ancestor classes in OWL file can be transformed into resources in resource space. The following are examples of two individuals:

```
<BMP rdf: ID="instance_0001">
   <NAME              rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
   cave305.bmp
   </NAME>
   <AUTHOR rdf:resource="Mr.Zhao"/>
   ……
</BMP>
<RESEARCHER rdf: ID="Mr.Zhao">
    <AGE          rdf:datatype="http://www.w3.org/2001/XMLSchema#unsignedInt">
    40</AGE>
    ……
</RESEARCHER>
```

The individual "instance_0001" is an instance of class “BMP”. It owns two properties “NAME” and “AUTHOR”. “Mr.Zhao” is also an individual which is instantiated from the class “RESEARCHER”. Here the input ancestor class is “File”. Only individuals inherited from it are regarded as resources. Therefore, the former individual is a resource. The latter indicates the value of property “AUTHOR” of “instance_0001”, so it will not be regarded as a resource in this example.

4.3 From Inheritance Hierarchy to Inheritance Axis

The ancestor classes inherently represent classification. This corresponds to the classification principle of RSM, so inheritance hierarchy of resources in OWL will be transformed into an axis named inheritance.

The process of forming inheritance axis consists of three steps:

(1) Parse the OWL file to find the subclasses and instances of every input ancestor class; and,
(2) Form a hierarchical structure like Fig. 4 according to their inheritance relationships: node represents class or instance and edge represents the inheritance relationship between classes or the instance relationship between instance and its class. So each edge starts from a class and ends at its super class or starts from an instance and ends at its class. The hierarchical structure is a top-down directed graph with ancestor classes at the top level and the instances (resources) at bottom level.
(3) Transform the structure into a tree or trees, which are taken as the coordinates.

In Dunhuang culture resource space, the class “File” is the input ancestor class to create the file resource space. It has four subclasses: “document”, “image”, “audio” and “video”, which also have their own subclasses. The subclasses of “document” are “PDF” and “TXT”. They indicate different types of files. Their declarations in Dunhuang OWL file are as follows:

```
<owl:Class rdf:ID="File"/>
<owl:Class rdf:ID="image">
      <rdfs:subClassOf rdf:resource="File"/>
 </owl:Class>
<owl:Class rdf:ID="JPEG">
       <rdfs:subClassOf rdf:resource="image"/>
 </owl:Class>
 <owl:Class rdf:ID="BMP">
       <rdfs:subClassOf rdf:resource="image"/>
 </owl:Class>
 <owl:Class rdf:ID="document">
      <rdfs:subClassOf rdf:resource="File"/>
 </owl:Class>
 <owl:Class rdf:ID="PDF">
      <rdfs:subClassOf rdf:resource="document"/>
 </owl:Class>
 <owl:Class rdf:ID="TXT">
      <rdfs:subClassOf rdf:resource="document"/>
```

```
</owl:Class>
<owl:Class rdf:ID="video">
    <rdfs:subClassOf rdf:resource="File"/>
</owl:Class>
<owl:Class rdf:ID="AVI">
    <rdfs:subClassOf rdf:resource="video"/>
</owl:Class>
<owl:Class rdf:ID="SWF">
    <rdfs:subClassOf rdf:resource="video"/>
</owl:Class>
<owl:Class rdf:ID="audio">
    <rdfs: subClassOf rdf: resource="File"/>
</owl:Class>
<owl:Class rdf:ID="MP3">
    <rdfs:subClassOf rdf:resource="audio"/>
</owl:Class>
<BMP rdf:ID="instance_0001">… </BMP>
<TXT rdf:ID="instance_0011">…</TXT>
<AVI rdf:ID="instance_0021">… </AVI>
```

The inheritance structure in Fig.4 includes the ancestor class, its offspring classes and these instances (resources). The top level is “File”. The second level includes “document”, “image”, “video” and “audio”. The lower level includes “PDF”, “TXT”, “JPEG”, “BMP”, “MP3”, “AVI” and “SWF”. The instances are resources at the bottom level. The resource “instance_0001”, “instance_0011” and “instance_0021” are instances of “BMP”, “TXT” and “AVI” respectively.

If the inheritance hierarchy is a tree or trees, it can be transformed into an axis to represent the category of resources. This axis is called inheritance axis. The elements in the hierarchy are the coordinates on inheritance axis except for the instances. If the hierarchy is a tree, there is only one node at the top level. This node will not be a coordinate on inheritance axis because it cannot classify resources. A resource’s coordinates on this axis is composed of its class and ancestor classes together. The coordinates at different levels represent different scales of classification. In Dunhuang resource space, the inheritance hierarchy is a tree, so it can be transformed into an inheritance axis directly named “Format”. As shown in Fig.5, the top level coordinates on this axis include “document”, “image”, “audio” and “video”. The second level coordinates are “TXT”, “PDF”, “JPEG”, “BMP”, “MP3”, “AVI” and “SWF”. The coordinate of “instance_0011” on this axis is “document.TXT”, where the dot separates coordinates on different scales.

OWL supports multiple-inheritance, that is, the inheritance structure is not a tree but a graph. The graph should be converted into tree(s) because the coordinates on an axis in RSM should be tree(s). Algorithm 1 converts an inheritance graph into tree(s). Multiple-inheritance indicates that one subclass inherits from two or more classes. It implies that the parent classes cannot classify the resources independently. So the algorithm eliminates the nodes of parent classes and linked edges directly from the hierarchy. The subclasses are reserved as the top level elements of the derived tree.

This algorithm guarantees that the output tree contains the resources and their parent classes at least. Here the function outdegree() and indegree() are used to get the out-degree and indegree of node in a graph respectively. setMark() and getMark() are for setting or getting markers of nodes. getParent() is for getting the parent class of a node in a graph. getUntreatedNumber() is for getting the number of untreated nodes in T.

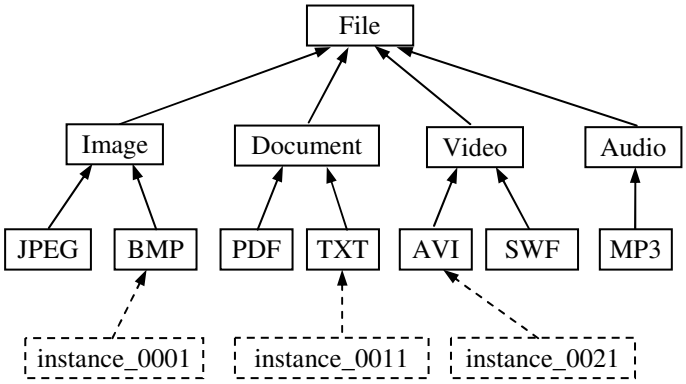


Fig. 4. The inheritance hierarchy of resources in Dunhuang application. The solid and dashed rectangles indicate classes and instances respectively. The solid and dashed arrows indicate inheritance relationships and instance relationships respectively.

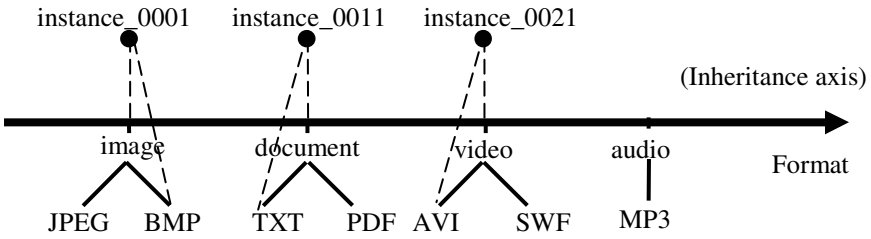


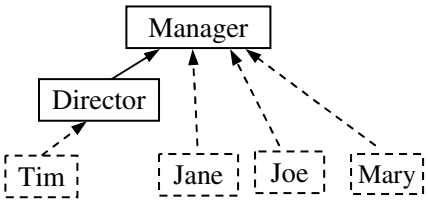
Fig. 5. The inheritance axis of Dunhuang resource space

```
Algorithm 1. void GraphToTree(Graph G , Tree T)
{/*convert a connected directed graph G into a tree(s)
T*/
  For every node
  {/*treat from the bottom level*/
    If( outdegree(node, G) = 0 ) {/*a bottom node*/
      Output node into T as a leaf;
      If( indegree(node, G) = 0 ) {
        Show message "error: an individual hasn't
class";
        Return;
      }
```

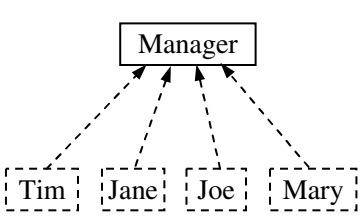
```
      Else if( indegree(node, G)=1 ) {/*uni-
inheritance*/
        setMark(node, T, treated);
        Output getParent(node,G) into T;
      }
    }
  }
  While( getUntreatedNumber(T)>0 ) {
    Get an untreated node from T;
    If( indegree(node, G) = 1 ) {/*qualified node*/
      If( getMark( getParent(node,G) ) != deleted ) {
        Output getParent(node, G) as parent of node
into T;
      }
    }
    Else if( indegree(node, G)>1 ) {/*multi-
inheritance*/
      For every ancestor of node in G {
        If( getMark(ancestor, G) != deleted ) {
          setMark( ancestor, G, deleted );
          Delete ancestor from T;
        }
      }
    }
    setMark( node, T, treated ); /*mark treated node*/
  }
}
```

In OWL, concrete classes may own subclasses and instances, but abstract classes can only have subclasses. For a concrete class that has both subclasses and instances, it is possible that the instances of the concrete class cannot be located by the subclass coordinates. For example in Fig.6 (a), the concrete class “Manager” has a subclass “Director” and three instances “Jane”, “Joe” and “Mary”. “Director” has its own instance “Tim”. If this structure is converted into coordinates of the inheritance axis, the coordinates include “Manager” and “Director” at two levels. The coordinate “Director” can only specify “Tim”, but cannot specify “Jane”, “Joe” or “Mary”. There are two strategies to deal with this kind of concrete classes: (1) discard its subclasses and combine the instances of subclasses into it; (2) add a new subclass for the concrete class, instances of the concrete class can be identified as instances of the subclass added. The former strategy weakens the classification semantics of resources but simplifies the process. The latter enhances the classification semantics.

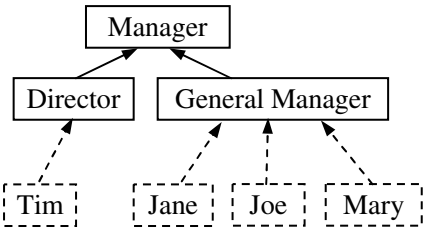
Algorithm 2 is to check and deal with concrete classes. The parameter bDiscard distinguishes the two strategies. If bDiscard = true, subclasses are discarded. Otherwise a new subclass is added whose name is provided by the parameter newClassName. Fig.6 (b) is the result when bDiscard=true. The subclass “Director” is deleted and its instance “Tim” becomes the instance of “Manager”. Fig.6 (c) shows the result when bDiscard=false. A new subclass named “General Director” is added with “Jane”, “Joe” and “Mary” as its instances. The result is transformed into the coordinates on the inheritance axis.



(a) Concrete class Manager with both subclass and instances



(b) Result of discarding subclasses



(c) Result of adding subclass

Fig. 6. An example of processing concrete class

```
Algorithm 2. Boolean CheckAndChangeConcreteClass (Class
conClass, Boolean bDiscard, String newClassName ) {
  If(conClass has both instances and subclasses) {
    If(bDiscard ) { /*discard subclasses*/
      For every subclass of conClass {
        Move its instances into conClass;
        Delete subclass;
      }
    }
    Else{ /*add a new subclass*/
      Create a new class named newClassName;
      Get all instances of conClass;
      Move the instances into newClassName;
      Add newClassName as a subclass of conClass;
    }
    Return true;
  }
  Else{/*need not be modified*/
    Return false;
  }
}
```

Then an inheritance axis is created according to inheritance relationships of resources and their ancestor classes. There is only one inheritance axis in the OWL-based resource space. The hierarchical coordinates provide users with multiple scale location according to application requirements.

4.4 From Properties to Property Axes

Properties in OWL are used to describe characteristics of classes and individuals. They can be adopted as classification principles of resources in RSM as well. If the domain of a property includes the ancestor classes, it can be transformed into a property axis. The property is called source property of the axis.

"DatatypeProperty" declares a property with one of data types, which come from RDF literals and XML Schema data types. The property value belongs to the specified datatype. A datatype property can be converted into an axis called datatype axis in OWL-based resource space. The axis is named after the property name and its coordinates include all elements within the property value range. If the range of datatype property is only specified as a kind of datatype without other restrictions, the elements in the range may be finite (such as "boolean" type) or infinite (such as "string" and "int" type). Because the coordinates on an axis must be finite, the unqualified property should classify their values into finite classes at first. Different datatypes use different strategies to classify its infinite elements into finite classes so as to ensure no intersection between classes. For example, "string" may be classified according to alphabet order. The classification strategy may contain hierarchical structure to classify resources with different scales. But the fixed classification approach classifies various resources into the same classes. Other classification methods in pattern recognition and text processing can be adopted to classify resources according to their characteristics. The restrictions of datatype property range are allowed in OWL. For convenience, Dunhuang OWL imports "xsp.owl" developed by Protégé to restrict data types. For example, Dunhuang resources have an "unsignedInt" type property "CaveNo" to specify which cave the resources reside in. Its declaration is as follows:

```
<owl:DatatypeProperty rdf:ID="CaveNo">
    <rdfs:domain>
        <owl:Class rdf:resource="File"/>
    </rdfs:domain>
    <rdfs:range>
        <rdfs:Datatype>
            <xsp:base
            rdf:resource="http://www.w3.org/2001/XMLSchema#unsignedInt"/>
            <xsp:minInclusive
            rdf:datatype="http://www.w3.org/2001/XMLSchema#unsignedInt">
            1</xsp:minInclusive>
            <xsp:maxInclusive
            rdf:datatype="http://www.w3.org/2001/XMLSchema#unsignedInt">
            900</xsp:maxInclusive>
        </rdfs:Datatype>
    </rdfs:range>
</owl:DatatypeProperty>
```

The domain of "CaveNo" is the class "File". The range of "CaveNo" is restricted from 1 to 900. Then the property can be transformed into an axis. Its coordinates are the elements within the property range. The resources and their property values are described as follows.

```
<BMP rdf: ID="instance_0001">
   <CaveNo>305</CaveNo>
   ……
  </BMP>
  <TXT rdf:ID="instance_0011">
   <CaveNo>220</CaveNo>
   ……
</TXT>
<AVI rdf: ID="instance_0021">
   <CaveNo>530</CaveNo>
……
</AVI>
```

The values of "instance_0001", "instance_0011" and "instance_0021" are 305, 220 and 530 respectively.

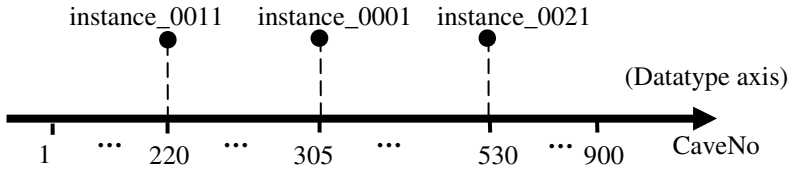


Fig. 7. The axis transformed by datatype property CaveNo

Fig. 7 is the axis named after the property "CaveNo". Its coordinates include all the cave numbers in Dunhuang from 1 to 900. A resource's coordinate on this axis is its property value. The coordinate of "instance_0001" on this axis is 305. So a datatype property can be transformed into a data type axis.

In OWL, an object property is a relation between two objects and declared by "ObjectProperty". An object property can be transformed into a homonymous axis, called object axis. Its coordinates consist of the ancestor classes of elements within the property's range and they are usually in inheritance hierarchy. A resource's coordinate on object axis is composed of the ancestor classes of its property's value. All the elements within the property's range, with their ancestor classes, form an inheritance hierarchy. The procedure of creating object axis is similar to that of creating inheritance axis. Algorithm 1 and algorithm 2 are also used to get a directed tree or trees. The output tree structure is converted into coordinates on object axis. For example, an object property "Content" is defined in Dunhuang OWL file as follows:

```
<owl:ObjectProperty rdf:ID="Content">
   <rdfs:domain rdf:resource="File"/>
   <rdfs:range rdf:resource="ContentClass"/>
</owl:ObjectProperty>
```

"Content" describes the content represented by Dunhuang resources. Its domain is "File" class and its range is "ContentClass". The values of resources on this property are described as follows:

```
<BMP rdf:ID="instance_0001">
    <Content rdf:resource="cave_305"/>
    …
 </BMP>
<TXT rdf:ID="instance_0011">
     <Content rdf:resource="story_1"/>
     …
</TXT>
<AVI rdf:ID="instance_0021">
      <Content rdf:resource="statue_530_2"/>
      …
</AVI>
```

The range of this property is declared as class “ContentClass”. Its subclasses are “painting”, “statue” and “architecture”. They also have their own subclasses, such as “flyer”, “story”, “separate”, “attached” and “cave”. “story_1”, “statue_530_2” and “cave_305” are the instances of “story”, “separate” and “cave” respectively. The values of resource “instance_0001”, “instance_0011” and “instance_0021” of property "Content" are “cave_305”, “story_1” and “statue_530_2” respectively. The inheritance hierarchy of "ContentClass" is given in Fig. 8.

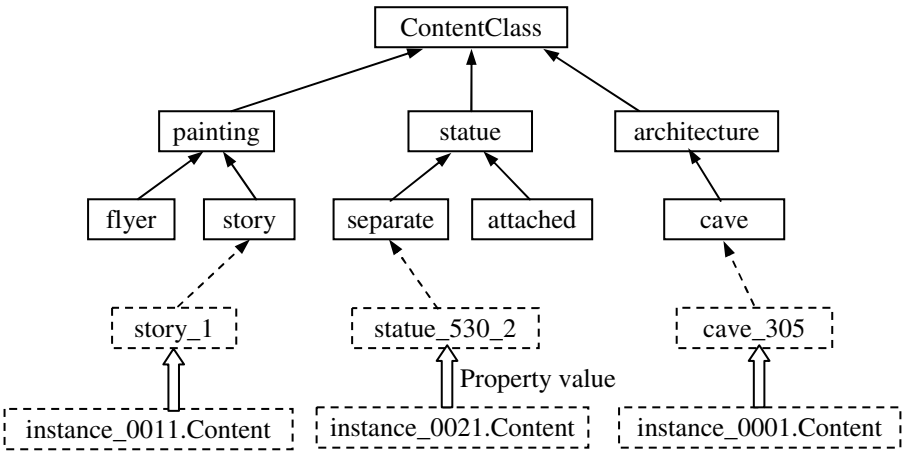


Fig. 8. The inheritance hierarchy of class “ContentClass” and the values of resources. Here the solid and dashed rectangles indicate classes and instances respectively. The solid arrows indicate the inheritance relations.

Based on the structure, an object axis named “Content” is created and shown in Fig.9. “ContentClass” is not transformed into a coordinate because there is only one element at the top level. The coordinates of “instance_0001”, “instance_0011” and “instance_0021” are “architecture.cave”, “painting.story” and “statue.separate” respectively.

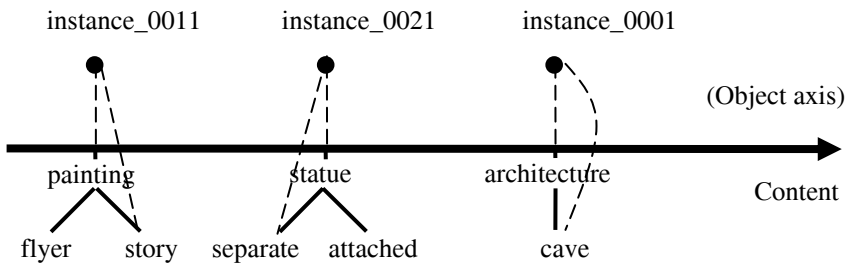


Fig. 9. Object axis derived from object property Content

In fact, not any property of ancestor classes in OWL should be transformed into a property axis in OWL-based resource space. For example, every Dunhuang resource has a property "NAME". Suppose that "NAME" is converted into an axis "NAME". Then each coordinate on it can only identify one resource if name duplication is prohibited. Axes generated from this kind of properties hardly classify resources efficiently. It is not an easy job to judge if a property should be transformed into an axis because it depends on the classification semantics represented by the property. It may need user's analysis and choice. The ratio of resource number to coordinate number on the axis can be used as a referenced principle: if the ratio is close to 1, the property should not be transformed.

4.5 Combination of Axes and Resources into Resource Space

Resources and axes derived from OWL are combined to form a coordinate system. Every resource has a location determined by their ancestor classes and property's values in this coordinate system. They are inserted into corresponding points in the space. A point in the space uniquely represents a set of resources. From the definition of resource space [16], this coordinate system constitutes a resource space. The structure of the Dunhuang OWL-based resource space is shown in Fig. 10. For simplification, only the coordinates on one layer in the coordinate hierarchy are shown.

There are three axes named "Format", "Content" and "CaveNo" respectively. The "Format" axis is the inheritance axis derived from the inheritance hierarchy of resources. The "CaveNo" is a datatype axis and directly comes from the homonymous datatype property. The object axis "Content" is transformed from the same name object property. Every resource in the space is specified by a tuple of coordinates. For instance, the resource "instance_0001" corresponds to the point (architecture, image, 305). That means the resource is an image file and describes the architecture of the 305^{th} cave.

The created resource space includes an inheritance axis and several property axes, which are transformed from the inheritance hierarchy and properties of resources respectively. The resources are derived from individuals and inserted into the resource space according to their ancestor classes and property values.

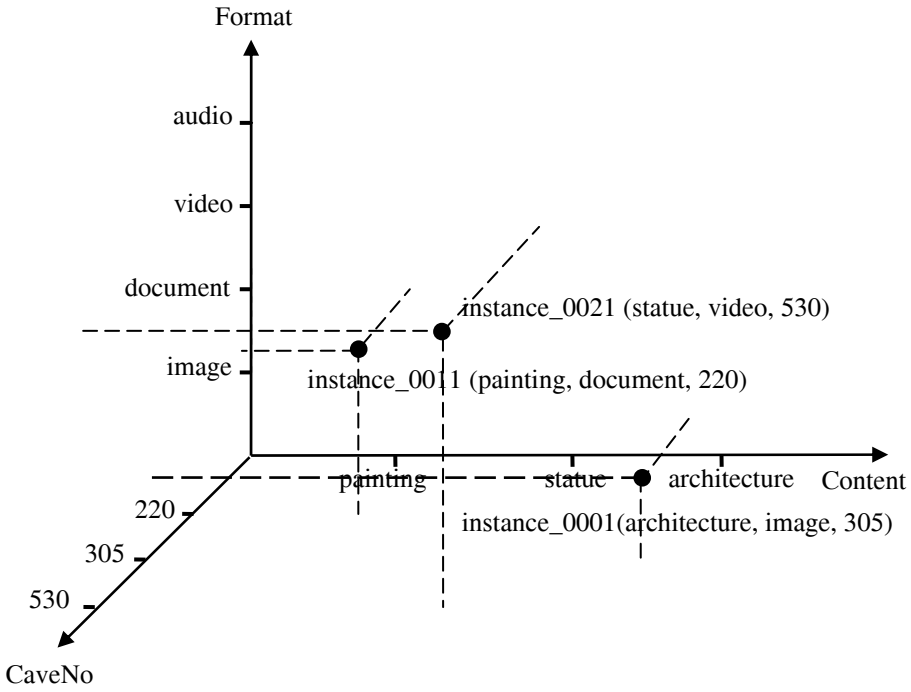


Fig. 10. The simplified resource space derived from the Dunhuang OWL file

5 Analysis of Normal Forms

Normal forms guarantee the correctness of operations on RSM [17, 20]. Normal forms of the resource space generated from OWL file are important for their successful application. Here we assume that the OWL file is well-defined (i.e., the file can represent domain knowledge correctly and clearly) so that the resource space can represent correct classification semantics. The coordinates on an axis can be hierarchical, but usually the coordinates at the same level can satisfy certain demand of application. The hierarchical coordinates can be mapped into flat coordinates by only projecting the same level coordinates onto the axis. So here only considers the flat case for simplification.

5.1 The First Normal Form

The first-normal-form (1NF) of resource space requires that there is no name duplication between coordinates at any axis. The 1NF can be easily checked by comparing all coordinates on one axis. The unqualified resource spaces can be upgraded to 1NF after combining the duplicate coordinates into one and the corresponding resources into one set.

A well-defined OWL file does not contain duplicated classes, instances and property values. So the coordinates consisting of classes, instances and property values at any axis should not be duplicated. Hence the OWL-based resource space satisfies 1NF.

5.2 The Second Normal Form

The second-normal-form (2NF) of a resource space is a 1NF, and for any axis, any two coordinates are independent from each other. The 2NF avoids implicit coordinate duplication, and prevents one coordinate from semantically depending on another. In the application, the implicit duplication and semantic dependence are concerned with the domain knowledge. Here the semantic independence means that a coordinate is not the synonym, abstract concept, specific concept, instance or quasi-synonym of another coordinate.

Since the synonymic classes, properties and individuals are already combined during preprocessing, there are no synonymic coordinates on the inheritance axis and property axes. In a well-defined OWL file, the abstract concept of a coordinate should be declared as its ancestor class. Because the hierarchical structure of coordinates is based on inheritance relations, the abstract concept and the coordinates are at different levels. So there is no abstract concept of a coordinate at the same level. The specific concept and instance of a coordinate should be its subclass and instance respectively. They are also at different levels in the hierarchical structure of coordinates. In order to avoid semantic confusions, the coordinates at different levels should not be used at the same time. During the procedure of creating inheritance axis, the multi-inheritance problem is solved. So every resource has a certain value on axis. The quasi-synonymic classes do not influence the resource classification. On the datatype axis, the coordinates are one type of values or their classification. The quasi-synonymic values cannot influence the resource classification because a resource's coordinate is a certain value. On an object axis, the resource coordinates are ancestor classes of their property values. The coordinates on object axis are similar to those on the inheritance axis. They can avoid classification confusion of resources. So there are no influential quasi-synonyms on any axis.

The 2NF avoids the intersection of resource sets on different coordinates. In the resource space created from a well-defined OWL file, the resources are classified clearly by the coordinates on any axis. So the coordinates on any axis are semantically independent. Generally, the classification confusion on axis implies that the OWL file contains some confusing description and it should be modified to prevent semantic confusion. In other words, a well-defined OWL file can be directly transformed into a resource space satisfying the 2NF.

5.3 The Third Normal Form

A 2NF resource space is 3NF if any two axes are orthogonal with each other [16]. From the generation process, we know that an OWL-based resource space contains one inheritance axis and several property axes. Then, we have the following lemmas:

Lemma 1. In the OWL-based resource space, any two axes are orthogonal, if and only if: (1) the inheritance axis is orthogonal to any property axes, and (2) any two property axes are orthogonal.

Lemma 2. The orthogonality between two axes is transitive, that is, if $X \perp X'$ and $X \perp X''$, then $X' \perp X''$ [17].

Lemma 3. In the OWL-based resource space, if the inheritance axis is orthogonal to any property axes, any two property axes are orthogonal with each other.

Proof. Let the inheritance axis be X^I, and, X_1^P and X_2^P be two arbitrary property axes. If the inheritance axis is orthogonal with any property axis, $X^I \perp X_1^P$ and $X^I \perp X_2^P$ hold. Because $X^I \perp X_1^P \Leftrightarrow X_1^P \perp X^I$ and according to Lemma 2, $X_1^P \perp X_2^P$ holds, i.e., two property axes are orthogonal. □

Theorem 1. If the OWL-based resource space is in 2NF and the inheritance axis is orthogonal with any property axes, the resource space satisfies 3NF.

Proof. From Lemma 3, if the inheritance axis is orthogonal with any property axes, we have: any two property axes are orthogonal.

From Lemma 1, we have: any two axes are orthogonal in the OWL-based resource space.

According to the definition of 3NF, the resource space is in 3NF. □

Lemma 4. For two axes X_i and X_j in a resource space, $X_j \perp X_i \Leftrightarrow R(X_j) = R(X_i)$ holds [20].

Theorem 2. In a 2NF OWL-based resource space, its inheritance axis is denoted as X^I. If $R(X^I) = R(X^P)$ holds for any property axis X^P, the resource space satisfies 3NF.

Proof. From Lemma 4, $R(X^I) = R(X^P) \quad X^I \perp X^P$.

Then the inheritance axis is orthogonal with any property axis.

According to Theorem 1, the resource space satisfies 3NF. □

Lemma 5. If a resource r owns the property P, then r can be represented by the property axis X^P transformed from P, that is, $r \in R(X^P)$.

Proof. According to the generation process of property axis, the coordinates on X^P may consist of three kinds of elements: all elements within the range, a classification of all elements in the range or the ancestor classes of all the elements within the range. Because r owns the property P, so the P's value of r is within the range, r has a coordinate on X^P. So $r \in R(X^P)$ holds. □

Theorem 3. If every property axis of the 2NF resource space *RS* is transformed from the common property (the property owned by all the ancestor classes of resources), the resource space *RS* satisfies 3NF.

Proof. Let E_R be the universal resources to be organized by *RS*, X^I be the inheritance axis and X^P be an arbitrary property axis. We can get $R(X^I) \subseteq E_R$ and $R(X^P) \subseteq E_R$.

For any resource r, we have $r \in E_R$.

(1) Since r is an instance of a class, r can find its ancestor class on X^I.

Then $r \in R(X^I)$ and $E_R \subseteq R(X^I)$ hold.

From $R(X^I) \subseteq E_R$, we can get $R(X^I) = E_R$.

(2) Since r is an instance of a class, it has the same properties of its ancestor class. P is a common property and owned by every ancestor class.

Then, we have: r must own P as its property.

From Lemma 5, $r \in R(X^P)$ holds.

Because $r \in E_R$ holds, we can get $E_R \subseteq R(X^P)$.

And from $R(X^P) \subseteq E_R$, then we have: $R(X^P) = E_R$ holds.
From (1) and (2), we get $R(X^I) = R(X^P)$.
Then according to Theorem 2, *RS* satisfies 3NF.

From Theorem 3, we know that if every axis in OWL-based resource space is created by a common property, then the resource space satisfies 3NF. Therefore the algorithm using this condition can generate a 3NF resource space.

6 Strategy and Discussion

Integration of OWL files developed by team members is very important in ontology engineering. Assume that OWL-file is the integration of OWL-file1 and OWL-file2 denoted as OWL-file=OWL-file1$\cup$ OWL-file2, and that RS, RS1 and RS2 are resource spaces created from OWL-file, OWL-file1 and OWL-file2 respectively. If the integration operation $\cup$ is defined according to the union of graphs, then it does not reduce resources, properties and classes, therefore RS_1 and RS_2 are the subspaces of RS (i.e., all resources, axes and coordinates in RS1 or in RS2 are also in RS). If there exist common axes between RS_1 and RS_2, then RS_1 and RS_2 can be integrated by join operation: $RS_1 \cdot RS_2$ [16, 17]. Since join operation does not increase any new axis, coordinate and resource, $RS_1 \cdot RS_2$ is also a subspace of RS. This tells us a strategy of transformation from OWL into resource space: *Integrate OWL files rather than resource spaces*, that is, select the integrated OWL file for transformation to reserve more semantics rather than select the individual OWL files for transformation and then integrate the created resource spaces.

The RSM can accurately locate resources and has a firm theoretical basis for ensuring the correctness of resource operations. A two-dimensional or three-dimensional resource space can be easily displayed, manipulated and understood in mental abstraction world. Higher-dimensional resource space needs to be split into several lower dimensional resource spaces by the split operation for display [16]. But its implementation depends on the underlying structure in the machine world.

The OWL is being widely accepted by researchers and ontology developers. There will be rich OWL-based ontologies, which are the basis of automatically generating the RSM. The OWL is not designed for human to read so it is hard for human to maintain it. The OWL needs to develop its theoretical basis.

Integrating OWL with RSM can obtain advantages and overcome shortcomings of both. One strategy is to place the RSM at the high level for efficient locating and effective management of resources and place the OWL description at the low level to provide ontology support. The underlying ontology supports the normalization of the RSM [16]. The join and merge operations of RSM support the management of multiple resource spaces which could be generated from the same OWL file.

7 Conclusion

This paper investigates the semantics of the interconnection environment, and proposes an approach to automatically create a resource space from a given OWL file, and analyzes the normal forms of the OWL-based resource space. This approach

can make use of existing ontology and relieve the dependence on developers' knowledge. The integration of RSM and OWL can obtain advantages of both. Strategies for transformation and integration are given. Ongoing work is the transformation between OWL and other forms of semantics like the semantic link network SLN [17].

Acknowledgement

The authors thank all team members of China Knowledge Grid Research Group (http://www.knowledgegrid.net) for their help and cooperation.

References

1. Assem, M., Menken, M. R., Schreiber, G., Wielemaker, J., Wielinga, B. J.: A Method for Converting Thesauri to RDF/OWL, International Semantic Web Conference, Hiroshima, Japan, (2004) 17-31.
2. Batini, C., Ceri, S., Navathe, S. B.: Conceptual Database Design: an Entity-Relationship Approach, Benjamin and Cummings Publ. Co., Menlo Park, California, 1992.
3. Berners-Lee, T., Hendler, J., Lassila, O.: The Semantic Web. *Scientific American*, 284(5) (2001) 34-43
4. Chen, P. P.: The Entity-Relationship Model, Towards a Unified View of Data, *ACM-Transactions on Database Systems*, 1 (1) (1976) 9-36.
5. Embley, D. W.: Object Database Development Concepts and Principles, *Addison Wesley*, 1997.
6. Gaševic, D., Djuric, D., Devedžic, V., Damjanovic, V.: Converting UML to OWL Ontologies, *Proceedings of the 13th International World Wide Web Conference*, NY, USA, (2004) 488-489.
7. Grønmo, R., Jaeger, M. C., Hoff, H.: Transformations between UML and OWL-S, *the European Conference on Model Driven Architecture Foundations and Applications (ECMDA-FA)*, Springer-Verlag, Nuremberg, Germany, November, 2005.
8. Gruber, T. R.: A Translation Approach to Portable Ontology Specifications, *Knowledge Acquisition*, 5 (2) (1993) 199-220.
9. Hepp, M.: A Methodology for Deriving OWL Ontologies from Products and Services Categorization Standards, *Proceedings of the 13th European Conference on Information Systems (ECIS2005)*, Regensburg, Germany, (2005), 1-12.
10. Kaljurand, K.: From ACE to OWL and from OWL to ACE, *The third REWERSE annual meeting*, Munich, March, 2006.
11. Marca, D., McGowan, C.: SADT: Structured Analysis and Design Techniques, McGraw-Hill, 1987.
12. Motik, B., Vrandecic, D., Hitzler, P., Sure, Y., Studer, R.: dlpconvert - Converting OWL DLP Statements to Logic Programs, *System Demo at the 2nd European Semantic Web Conference,* Iraklion, Greece, May, 2005.
13. Ng, P. A.: Further Analysis of the Entity-Relationship Approach to Database Design, *IEEE Transaction on Software Engineer*, 7(1) (1981) 85-99.
14. Neches, R., Fikes, R. E., Gruber, T. R., Patil, R., Senator, T., Swartout, W.: Enabling Technology for Knowledge Sharing, *AI Magazine*, 12 (3) (1991) 36-56.
15. Teorey, T., Yang, D., Fry, J.: A Logical Design Methodology for Relational Databases Using the Extended Entity-Relationship Model, *ACM Computing Surveys*, 18 (2), June, 1986.

16. Zhuge, H.: Resource Space Grid: Model, Method and Platform, *Concurrency and Computation: Practice and Experience*, 16 (14) (2004) 1385-1413
17. Zhuge, H.: The Knowledge Grid, World Scientific, Singapore (2004)
18. Zhuge, H.: Resource Space Model, Its Design Method and Applications, *Journal of Systems and Software*, 72 (1) (2004) 71-81
19. Zhuge, H., Xing, Y.: Integrity Theory for Resource Space Model and Its Application, Keynote, *WAIM2005*, LNCS 3739, (2005) 8-24
20. Zhuge, H., Yao, E., Xing, Y., Liu, J.: Extended Normal Form Theory of Resource Space Model, *Future Generation Computer Systems*, 21 (1) (2005) 189-198.
21. Zhuge, H.: The Open and Autonomous Interconnection Semantics, Keynote at 8th International Conference on Electronic Commerce, August 14-16, 2006, Fredericton, New Brunswick, Canada.