Chapter 3

The OntOAIr method: construction of ontologies

As discussed in the previous sections, ontologies are manually created by domain experts or semi-automatically by means of ontology learning methods. In both cases, they specify semantic relationships between the terms in a lexicon. Ontologies can also be derived from hierarchical collections of documents such as the Yahoo! Directory [65] and the Open Directory Project [54].

This section describes the OntOAIr method, (Ontologies from Open Archives Initiative Repositories to Support Information Retrieval), a method that we first proposed in [46] to semi-automatically construct ontologies from data providers called *ontologies of records*. These are hierarchical structures formed by disjoint groups of records. Ontologies of records have two main uses: (1) To organize records based on their content; and (2) To support information retrieval tasks from multiple data providers.

The process of constructing ontologies of records consists of four main tasks: harvesting, representation, clustering and formalization. The harvesting task obtains the documents from the digital collections. The representation task constructs a vectorial

representation for each harvested document. The clustering task applies an exclusive hierarchical algorithm to the vectors in order to produce a tree of clusters. The formalization task transforms the tree of clusters into a lightweight ontology. Figure 3.1 shows the sequence of these tasks, which are described with further detail in the following sections.

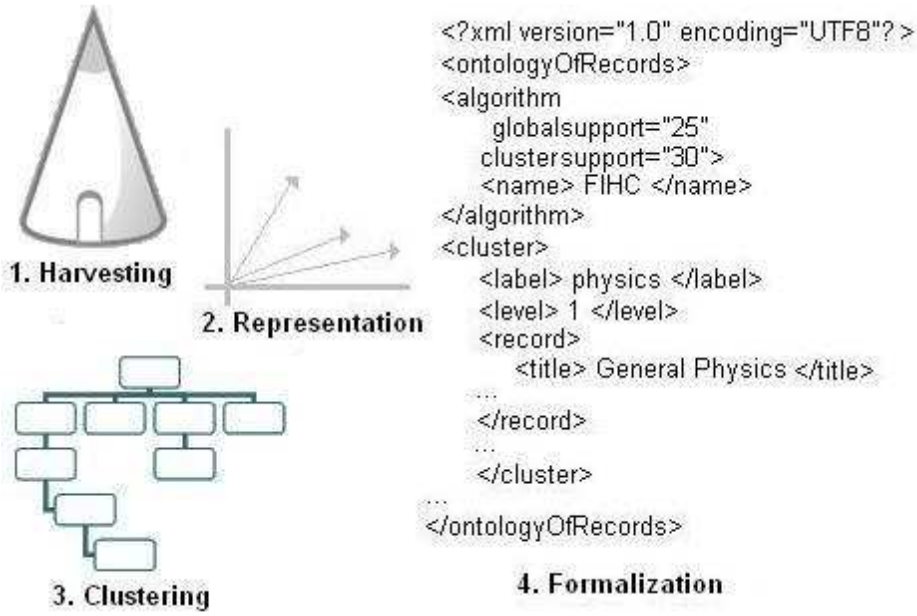

Figure 3.1: Main tasks to construct ontologies of records

3.1 Harvesting

Harvesting uses the *request verbs* listed in Table 1.2 to obtain metadata records from data providers. This is a repetitive, time-consuming task because data providers typically listed hundreds or thousands of records that need to be transmitted throught the network; for this reason, in our work, harvesting has been delegated to software entities

called *harvester agents*. This notion was first discussed in our early work [44], which included a detailed description of these agents.

Harvester agents are modeled with Gaia [64], a general purpose methodology that supports the analysis and design of multi-agent systems. Gaia is based on the roles that an agent plays in a system to identify types and relationships between agents.

A role is defined by responsibilities, permissions, activities and protocols. Responsibilities determine the functionality of the role. Permissions refer to the rights associated with every role. Activities are the computations that may be carried out by the agent without interacting with other agents, whereas protocols define the ways an agent interacts with other agents.

In Gaia, there are two types of responsibilities: existence and safety. Existence responsibilities describe states of affairs that an agent must bring about; while safety responsibilities inform that an acceptable state of affairs is maintained across all states of execution.

Tables 3.1 and 3.2 show the roles of our harvester agents. The first role uses the `Identify` request verb and the second role the `ListRecords`, `ListIdentifiers` and `GetRecord` verbs.

A complexity analysis of harvesting offers interesting challenges because it depends on external factors such as network overhead and availability of data providers.

Role: Forms a registry of data providers		
Description Extracts information to identify a data provider.		
Protocols and Activities queryDataProvider, registerDataProvider		
Permission		
read		
	infoDataProvider	*// Extracts information to identify a data provider*
write		
	database	*//Once the identification of a data provider is gotten, it is stored in the database*
Responsibilities Existence • Registers a data provider in the database Safety • Checks that the data provider has not been registered in the database • Checks the completeness of data provider information • Makes a request to the data provider to verify its availability		

Table 3.1: Scheme of the role to form a registry of data providers.

Role: Harvests records	
Description	
Harvests records from a data provider.	
Protocols and Activities	
getRecords	
Permissions	
read	
getAddress	*// Gets the address of a data provider*
harvestRecords	*// Harvests all the records from a data provider*
write	
storeRecords	*//Stores the harvested records in a temporal database*
Responsibilities	
Existence	
• Gets the update date of harvested records	
Safety	
• Compares the update dates of harvested records with the previous version of the ontology of records of the data provider	

Table 3.2: Scheme of the role to harvest records.

Label	Definition	Comment
`dc:description`	An account of the resource	Description may include but is not limited to: an abstract, a table of contents, a graphical representation, or a free-text account of the resource
`dc:title`	A name given to the resource	It will be a name by which the the resource is formally known

Table 3.3: DC elements that have content information of records

3.2 Representation

Once records from data providers have been harvested, simplified representations of records are produced. As mentioned earlier, unqualified Dublin Core (DC) is the default metadata format for OAI-PMH Version 2.0. In our work, content information of records is extracted from the elements `dc:title` and `dc:description`. They contain the title and the description of a resource, respectively. `dc:title` and `dc:description` have just one value.

Table 3.3 shows the definition and a brief comment associated to DC elements according to the DC Metadata Initiative [13]. Note that these elements are not mandatory.

DC elements contain free text. Before processing, any case sensitivity in the data is removed. Then, feature vectors are generated. A *feature vector* is a simplified representation of a record formed by keywords (all terms other than stop-words extracted from DC elements) and weights (numeric values that represent the relevance of keywords). The name is adopted from [16].

Our work adds two new elements to a feature vector: an identifier to discriminate between records and a URL with the location of its data provider. Both elements

Table 3.4: Example of a feature vector

Identifier	oai:thesisUDLAP:98	
URL	$http://ict.udlap.mx:9090/Tales$	
Keyword	**TF**	**IDF**
digital	1	10.397
libraries	3	9.133
reference	1	6.089
services	1	7.783

uniquely identify a record in OAI; they do not add content information. Table 3.4 shows a typical feature vector. The weights reflect that some words are better at discriminating between records than others. A TF-IDF factor (term frequency - inverse document frequency) is used as the weight of keywords.

Feature vectors allow users to have a compact view of a data provider. They can be stored in text files or in a database to be processed afterwards.

3.3 Clustering: adaptation of FIHC

The clustering consists in choosing an exclusive hierarchical algorithm to obtain groups of similar feature vectors. We first suggested the use of the FIHC algorithm in [46] for this purpose.

The selection of FIHC algorithm is based on the following hypothesis: "if a group of documents refers to the same topic, the documents would share a set of terms" [16]. We believe that the terms of cluster labels can be used to construct a vocabulary to describe the main topics of a document collection.

We implemented FIHC algorithm as proposed by [16], however, we experimentally determined that pruning the tree of clusters does not achieve a better clustering. Pruning the tree corresponds to the steps 5, 6 and 7 of FIHC algorithm described in Section 2.1.2.

Child pruning is addressed to reduce the width of the tree of clusters until a user-specified number of clusters is reached. From our point of view, this process does not contribute to improve the cluster accuracy since the number of main clusters in document collections is unknown.

Sibling pruning is addressed to reduce the depth of the tree of clusters considering only the non-leaf nodes at level two or greater. Since the F and entropy measures take the nodes at level one into account, under these measures, clustering accuracy would not be affected by the elimination of this process.

Therefore, we propose an adaptation of FIHC algorithm where the child pruning and the sibling pruning are removed from the construction of the tree of clusters.

3.4 Formalization

The formalization task refers to the representation of the tree of clusters in a machine-accessible language. This section presents traditional representations of trees of clusters and then it describes representations based on ontology markup languages to construct the ontologies of records.

Considering our goals, the XML representation of DocCluster lacks a structure to validate the tree of clusters in such a way that it can be used by parsers and other XML processing software. For this reason, we have constructed a DTD that represents a generic document hierarchy. This DTD was first proposed in [48] and is the basis to construct the XML mapping of ontologies of records.

3.4.1 XML mapping

Figure 3.2 shows the elements of our previous work about generic document hierarchy. A document hierarchy is the output of an exclusive hierarchical clustering algorithm. The algorithm uses input parameters represented as attributes. The `cluster` element is formed by `label`, `level` and `document` elements. The `num_documents` attribute stores the number of documents of a cluster.

We adapt the elements of Figure 3.2 to construct the structure of an ontology of records. Conceptually, an ontology of records is divided into two main sections:

- *Cluster hierarchy.* The central concept here is *cluster*, which is intended to capture the notion of anything that involves a set of records
- *Record description.* The central concept of this section is *record*, which represents a document in the OAI repositories

Table 3.5 shows the DTD for an ontology of records. This DTD was first proposed in [41].

Table 3.5: DTD for an ontology of records

```
<!ELEMENT ontologyofrecords
(algorithm, cluster+)>
<!ATTLIST ontologyofrecords
date CDATA #REQUIRED >

<!ELEMENT algorithm EMPTY>
<!ATTLIST algorithm
name CDATA #FIXED ‘‘FIHC’’
globalsupport  CDATA #REQUIRED
clustersupport CDATA #REQUIRED>

<!ELEMENT cluster
(label, level, record*)>
<!ELEMENT label (#PCDATA)>
<!ELEMENT level (#PCDATA)>

<!ELEMENT record
(title, subject?, description?,
 identifier, url, dataprovider,
 metadataformat,  datestamp>
<!ELEMENT title (#PCDATA)>
<!ELEMENT subject (#PCDATA)>
<!ELEMENT description (#PCDATA)>
<!ELEMENT identifier (#PCDATA)>
<!ELEMENT url (#PCDATA)>
<!ELEMENT dataprovider (#PCDATA)>
<!ELEMENT metadataformat (#PCDATA)>
<!ELEMENT datestamp (#PCDATA)>

<!ENTITY generatedBy ‘‘OntoSIR 2.1’’>
```

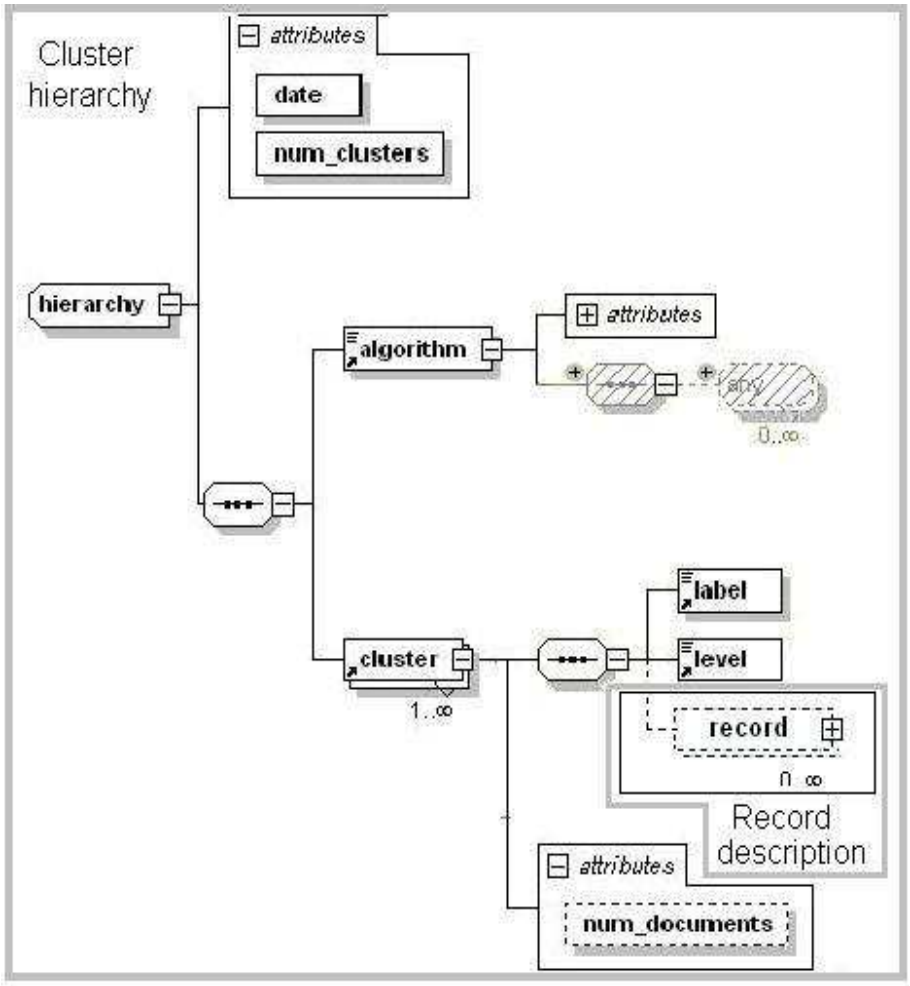

Figure 3.2: Elements of a generic document hierarchy

The DTD of Table 3.5 is interpreted as follows:

1. The `date` attribute and the `algorithm` element describe metadata of the ontology of records

2. At least one `cluster` element is needed

3. The `record` element represents a resource in the OAI Initiative

Table 3.6: Data types used in the XML Schema of an ontology of records

Name	Component Type	Data Type
cluster_support	attribute	`xsd:integer`
date	attribute	`xsd:date`
datestamp	element	`xsd:date`
global_support	attribute	`xsd:integer`
level	element	`xsd:integer`
url	element	`xsd:anyURI`

4. Each `cluster` has a `label`, a `level` and zero or more `records`

5. The `subject` and `description` elements are optional. The rest of the elements are required.

The use of `title`, `subject`, `description` and `identifier` elements is defined by DC (see Table 3.3 and 3.4). The `url`, `dataprovider`, `metadata format` and `datestamp` elements are used for identification purposes in OAI. The remaining elements correspond to the concepts specified by the FIHC algorithm and by the generic document hierarchy.

A disadvantage of using a DTD to validate ontologies of records is that all the elements are treated as strings of characters. An XML Schema enables the inclusion of data types. It also makes it possible to use namespaces to redefine or to extend the generated clusters by means of parsers and applications that enhance data validation. Table 3.6 shows the data types used in the XML Schema of an ontology of records [1] . The rest of the elements are of `xs:string` type. The XML implementation of the ontology of Figure 2.2 uses the XML Schema of the Appendix A.

Representing an ontology of records in XML inherits all the advantages of the lan-

[1] Namespace for an ontology of records.
$< http://www.utm.mx/\ sigepr/recursos/ontologyofrecords.xhtml >$.
June 16th 2008

guage itself such as its simplicity and readability. An ontology thus represented can be reused in a variety of tasks and stored directly in a file or even in enabled or native XML databases. However, there are also some shortcomings. For example, there is no standard way to assign meaning to nested tags, thus leading to ambiguity when interpreting XML documents. From an ontological point of view, the main drawback is that the semantics of XML documents is accessible to humans but not to machines.

In order to add machine-accessible semantics to ontologies of records, we base our implementation on Resource Description Framework Schema (RDF Schema). RDF-Schema is a semantic extension of Resource Description Framework (RDF), which is a framework to depict web metadata [18]. The following section describes the RDF Schema implementation of ontologies of records.

3.4.2 RDF mapping

RDF is the World Wide Web Consortium (W3C) standard to encode knowledge. It is a domain independent data model that groups *statements* (subject-predicate-object triples) and also an ontology markup language [19], [23].

Semantic nets (node and edge labeled directed graphs), are used to depict the structure of RDF-documents, where an edge and its connected nodes form a statement. Nodes represent concepts, instances of concepts or property values, whereas edges correspond to properties or relationships between concepts. Figure 3.3 exemplifies a semantic net for an ontology of records.

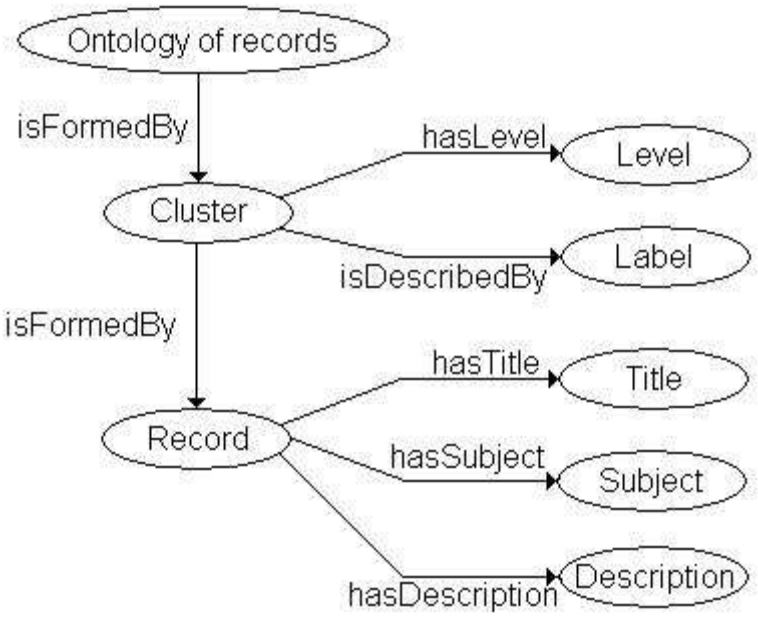

Figure 3.3: A semantic net for an ontology of records

Commonly, XML is used to represent RDF syntax. In this case, RDF data types are predefined by an XML Schema. We adopt this convention and transform XML ontologies into RDF-documents. The expressivity of RDF-documents is greater than XML ontologies in the sense that relationships between elements can be defined. However, the semantics of RDF documents is not machine-accessible. RDF Schema is required for this purpose.

RDF Schema [7] is a language for declaring basic classes and types to describe the terms used in RDF documents. RDF Schema provides modeling primitives to define the relationships between attributes and resources. It extends RDF with frame-based primitives.

By using RDF Schema, it is possible to define vocabularies in RDF documents,

specify properties and restrict its range to model assumptions about any particular application domain. Thus, RDF Schema offers a unique interpretation of ontologies of records that makes semantic information machine-accessible. The RDF mapping of an ontology of records was first proposed in [43].

Tables 3.7 and 3.8 show some relationships between ontology elements using RDF Schema. The definition of a class by each element allows us to impose restrictions on what can be stated in RDF ontologies that use the RDF Schema. This mapping was first discussed in our early work [47].

In RDF Schema, properties are defined globally, that is, they are applied to all classes. Therefore, to construct valid RDF ontologies of records, it was necessary to rename the property `isFormedBy` between `cluster` and `record` nodes of Figure 3.3 as `contains`. The RDF implementation of the ontology of Figure 2.2 uses the RDF schema of the Appendix B.

The expressiveness of RDF Schema is limited mainly because only binary relationships can be represented. Thus it is considered a primitive ontology language [19]. In order to extend the expressivity of RDF Schema, we propose the use of the Web Ontology Language (OWL) to explore another mapping of ontologies of records. This mapping is described next.

Table 3.7: Part 1. RDF Schema implementation of an ontology of records

```
<?xml version=‘‘1.0’’?>
<rdf:RDF
    xmlns:rdf=‘‘http://www.w3.org/1999/02/22-rdf-syntax-ns#’’>
    xmlns:rdfs=‘‘http://www.w3.org/2000/01/rdf-schema#’’>

<rdfs:Class rdf:ID=‘‘OntologyOfRecords’’>
</rdfs:Class>
<rdfs:Class rdf:ID=‘‘Cluster’’/>
<rdfs:Class rdf:ID=‘‘Label’’/>
<rdfs:Class rdf:ID=‘‘Level’’/>
<rdfs:Class rdf:ID=‘‘Record’’/>
<rdfs:Class rdf:ID=‘‘Title’’/>
<rdfs:Class rdf:ID=‘‘Subject’’/>
<rdfs:Class rdf:ID=‘‘Description’’/>

<rdf:Property rdf:ID=‘‘isFormedBy’’>
  <rdfs:comment>
     It relates clusters with an ontology of records
  </rdfs:comment>
   <rdfs:domain rdf:resource=‘‘#OntologyOfRecords’’/ >
   <rdfs:range rdf:resource=‘‘#Cluster’’”/ >
</rdf:Property>

<rdf:Property rdf:ID=“contains””>
  <rdfs:comment>
     It relates clusters with records
  </rdfs:comment>
   <rdfs:domain rdf:resource=‘‘#Cluster’’/ >
   <rdfs:range rdf:resource=‘‘#Record’’/ >
< /rdf:Property>
```

Table 3.8: Part 2. RDF Schema implementation of an ontology of records

```
<rdf:Property rdf:ID=‘‘ isDescribedBy ’’>
   <rdfs:domain rdf:resource=‘‘#Cluster’’/ >
   <rdfs:range rdf:resource=‘‘#Label’’/ > </rdf:Property>

<rdf:Property rdf:ID=‘‘ hasLevel ’’>
 <rdfs:domain rdf:resource=‘‘#Cluster’’/ >
 <rdfs:range rdf:resource=‘‘#Level’’/ > < /rdf:Property>

<rdf:Property rdf:ID=‘‘ hasTitle ’’>
   <rdfs:domain rdf:resource=‘‘#Record’’/ >
   <rdfs:range rdf:resource=‘‘#Title’’/ > < /rdf:Property>

<rdf:Property rdf:ID=‘‘ hasSubject ’’>
   <rdfs:domain rdf:resource=‘‘#Record’’/ >
   <rdfs:range rdf:resource=‘‘#Subject’’/ > < /rdf:Property>

<rdf:Property rdf:ID=‘‘ hasDescription ’’>
   <rdfs:domain rdf:resource=‘‘#Record’’/ >
   <rdfs:range rdf:resource=‘‘#Description’’/ > < /rdf:Property>

<rdf:Property rdf:ID=‘‘date’’>
   <rdfs:domain
     rdf:resource=‘‘#OntologyOfRecords’’/ >
   <rdfs:range
   rdf:resource=‘‘#&rdf;Literal’’/ > </rdf:Property>

</rdf:RDF>
```

3.4.3 OWL mapping

Since ontologies of records represent large document collections, an OWL implementation of ontologies might have reasoning support to check consistency and at the same time, extending the expressivity of RDF ontologies.

OWL was created by the W3C Web Ontology Working Group. It is built upon RDF and RDF Schema and it is based on the earlier languages OIL and DAML+OIL. DAML+OIL is briefly defined as RDF Schema with frame based knowledge representation primitives without the reification mechanism (reification in RDF consists in transforming the value of a property into a statement). DAML+OIL provides description logics extensions of RDF Schema [18].

OWL is divided into three sublanguages [40],[18]:

- *OWL Full*: This sublanguage offers the maximum expressiveness; it is fully upward compatible with RDF. However, software tools do not guarantee complete reasoning for all the characteristics (OWL Full is undecidable)
- *OWL Description Logics (DL)*: This sublanguage is oriented to cases where the maximum expressiveness is required without losing the computational completeness of reasoning systems. OWL DL includes the complete OWL vocabulary. OWL DL reasoning can be supported via Description Logics (DL) or First Order Logic (FOL)
- *OWL Lite*: This sublanguage is used to represent hierarchies and simple constraints. For example, while OWL Lite supports cardinality constrainsts, it only

permits cardinality values of 0 or 1

We use OWL DL which enables efficient reasoning about the knowledge of ontologies of records and maintains compatibility with RDF ontologies. Part of the knowledge about an ontology of records that can be represented in OWL DL is described in the following sentences:

- `OntologyOfRecords`, `Cluster`, `Level`, `Label`, `Record`, `Title`, `Subject`, `Description`, `Identifier`, `URL`, `DataProvider`, `MetadataFormat`, and `Datestamp` are OWL classes of an ontology of records
- An ontology of records is formed by clusters,
  (a `cluster` *is part of* an `ontology of records`)
- A `cluster` *contains* `records` (a `record` *is contained* in a cluster)
- A cluster *is described* by a label (a label *describes* a `cluster`)
- A `cluster` *has* a `level` in the ontology
- A `record` *has* `title`, `subject`, `description`, `identifier`, `URL`, `DataProvider`, `MetadataFormat`, and `Datestamp` elements
- The properties *isFormedBy* and *contains* are transitive

In the above list, the words in italics are names of properties, and sentences in parenthesis represent the inverse of a property.

Tables 3.9 shows an excerpt of the ontology of records in OWL language. As illustrated in the table, these mappings have sections to declare namespaces, create classes

Table 3.9: Excerpt of OWL model of an ontology of records.

```
<?xml version="1.0"?> <rdf:RDF
  xmlns:rdf=''http://www.w3.org/1999/02/22-rdf-syntax-ns#''
  xmlns:xsd=''http://www.w3.org/2001/XMLSchema#''
  xmlns:rdfs=''http://www.w3.org/2000/01/rdf-schema#''
  xmlns:owl=''http://www.w3.org/2002/07/owl#''>

<owl:Ontology rdf:about=''''>
  <rdfs:label>Ontology of records</rdfs:label>
  <owl:versionInfo>August 2007 2.1</owl:versionInfo>
</owl:Ontology>
<owl:Class rdf:ID=''OntologyOfRecords''/>

<owl:Class rdf:ID=''Cluster''>
  <rdfs:comment>Clusters form an ontology of records</rdfs:comment>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:allValuesFrom>
   <owl:Class rdf:about=''#OntologyOfRecords''/>
      </owl:allValuesFrom>
      <owl:onProperty>
   <owl:ObjectProperty rdf:ID=''isFormedBy''/>
      </owl:onProperty>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

and define properties.

The complete model for an ontology of records in OWL DL is included in Appendix C. The model comprises sections that declare namespaces, create classes and define properties. An example of a file of instances can be found in Appendix D.

To verify the feasibility of the OntOAIr method, we propose two applications that use the ontologies of records to support similarity-based retrieval tasks: a keyword-

based retrieval model and (2) an ontology-based exploration model. The next chapter describes these models.