

Semantic Data Integration in Hierarchical Domains

Isabel F. Cruz and Afsheen Rajendran, *University of Illinois at Chicago*

Data integration is a central issue within Semantic Web research, especially when it comes to developing standards and techniques that facilitate data integration without user intervention.¹ This sort of integration could have wide-ranging applicability in data-intensive applications such as search engines, data mining systems,

and data warehousing. A chief requirement of data integration systems is that the differences in the syntax and semantics of the underlying data sources should be hidden from the user.²⁻⁵ However, data sources are often independently created and do not easily interoperate with other sources.

The rise of XML as a data interchange standard has led to a new era in which research has focused on semistructured data.⁶ Although XML has provided a single interchange format, different users can model the same data in different ways, which can lead to heterogeneities at various levels, including the semantic level. Semantic heterogeneities can arise from entities being perceived differently. For example, an agricultural expert perceives waterways to be a source of irrigation, while a transportation expert perceives them as a mode of transportation. Such perceptual differences manifest themselves as heterogeneities in the data models these experts develop. These differences cannot be resolved without knowledge of the models of the underlying disciplines.

Statewide geographic information applications provide for an application domain whose need for data integration is compelling, especially if the data maintained by hundreds of autonomous government agencies is to be accessed uniformly. Geographic data is usually hierarchically organized. For example, counties contain municipalities, and wards contain precincts. However, different agencies use different hierarchies for representing data. So, the problem is how to handle the different classification schemes and the different types of relations between entities,

within the constraints imposed by the semantics of the data usage.⁷

This article describes one approach for handling semantic data integration problems in hierarchical domains. It also describes a declarative approach for specifying pairwise mappings between a centrally maintained ontology and each local data repository maintained by an autonomous agency. In this context, we outline a method for specifying the mappings' semantics and encoding them to resolve heterogeneities. We focus on XML-based applications in which entities in the centrally maintained ontology are hierarchically related to those in the local data repositories. We have implemented and successfully tested our approach in two different real-world applications: one involves querying the results of the US presidential election and the other involves querying a state's land use patterns.

Data integration systems

Data integration systems must have components for handling the mappings between entities in a global schema (called the *ontology*) and the local data sources. These systems must also have components for processing user queries expressed on the global schema.⁸ Typically, data integration systems convert the queries into subqueries expressed in terms of the entities in the local data sources using the mappings defined between the entities. After handling the differences in representation, the system merges the results of the subqueries and then displays those results to the user.

Generally, there are two approaches for specifying

A major challenge in building the Semantic Web is resolving differences among heterogeneous databases. This article outlines an XML-oriented approach to resolve semantic heterogeneities.

the mappings between the global schema and the local data sources. The *global-as-view* (GAV) approach associates every entity in the global schema with a view of the data sources. This approach ties the meaning associated with the entity to the local data. The *local-as-view* (LAV) approach defines the sources as views over the global schema. This approach specifies the global schema independently from the sources.

In the LAV approach, you can easily maintain and extend the system. When you need to add a new source, you need only to provide its definition without changing the global schema. In the GAV approach, system maintenance is more difficult. Adding a new source might require changing the definition of the entities in the global schema, but the query-processing techniques in the LAV approach are more sophisticated than in the GAV approach.⁸

We prefer a local-as-view approach because it gives us the ability to add or update new local data sources with minimal effort. In particular, we do not need to combine answers from different data repositories to produce results for a particular region. For example, in our election results application (described in more detail later), each local data repository is queried on the votes for a particular candidate, and each repository will return its own votes for that candidate, as associated with that particular geographic region.

In our case, the complexity of answering the queries is related to the semantic connections that must be established between the different classification hierarchies, which will drive the querying process between the ontology and the local schemas. An expert team, which develops the ontology, knows the application domain but is unaware of the local data sources' organization. Local experts develop the mappings between the ontology and the local data repositories.

The data collected by an agency might add levels of classification to those already in the global schema, reflecting the agency's primary area of interest. For example, a county whose main occupation is agriculture will have more categories of agricultural land use than the global schema drawn up for the state. Such differences in data resolution can be handled by storing that information in the ontology. Such an approach departs from a pure LAV approach because the team of experts integrates information from the local repositories into the ontology to improve the query answers.

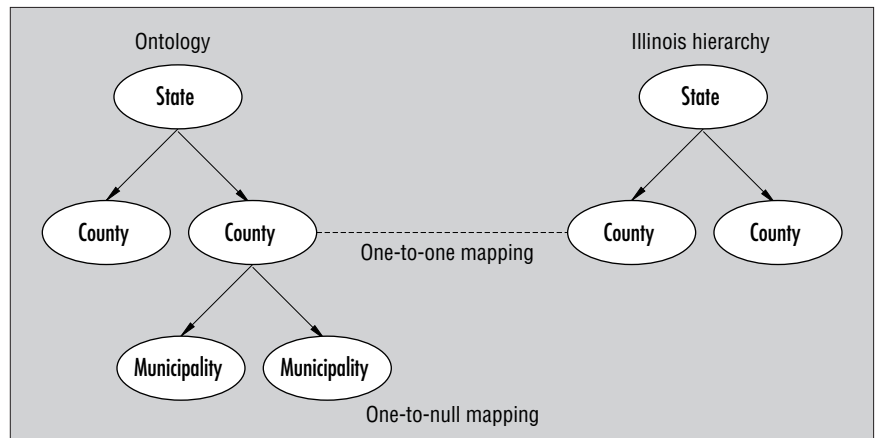


Figure 1. The one-to-null mapping type.

Ontologies and agreement documents

Because an ontology serves as the common source of understanding and as the basis for providing interoperability, it is independent from its representation in the domain. Our approach employs hierarchical ontologies, which it represents as XML documents. Information about the differences in hierarchies and types of relationships between two hierarchies are stored in XML documents called *agreement documents*. Each local data repository has a corresponding agreement document that acts as a wrapper.

We represent the relationships between equivalent hierarchy entities as mappings. One key responsibility of the local experts is to identify the types of mappings needed to map the entities in the ontology to those in the local data repository. To maintain a uniform query interface, the ontology's entities must be mapped to equivalent entities in the local data repository. The mappings fall broadly into these types:

- *One-to-one*. The entity in the ontology has an equivalent entity in the local data repository.
- *One-to-null*. The entity in the ontology has no equivalent entity in the local data repository. User queries for this entity cannot be supported.
- *Parent-children*. The entity in the ontology is equivalent to a collection of entities following a parent-children relationship. In this case, only the parent entity must be mapped to the corresponding entity. This mapping type uses the information about the hierarchy to simplify mappings by allowing just one entity to be mapped instead of a collection of entities.

- *One-to-many*. The entity in the ontology is equivalent to a collection of entities in the local data repository. The entities in the collection do not follow a parent-children relationship.
- *Many-to-one*. The entity in the local data repository is equivalent to a collection of entities in the ontology. The entities in the collection do not follow a parent-children relationship.

In our election results application, we identified the one-to-null and parent-children mappings. In the hierarchy for the state of Illinois election results, the *municipality* entity does not exist, which means that no equivalent exists for that ontological entity. The mapping type chosen for that match is one-to-null (see Figure 1).

While Illinois does not have municipalities, the *municipality* entity exists in the state of Wisconsin hierarchy. In addition, each municipality can have multiple *ward group* entities below it in the hierarchy. We mapped the ontological entities *state*, *county*, and *municipality* to their equivalent one-to-one entities in the Wisconsin hierarchy. A municipality and its constituent ward groups follow a parent-children relationship, so we chose the parent-children type (see Figure 2).

Representing information about a local data repository's hierarchy in XML is relatively straightforward. However, representing the mappings between equivalent entities in the ontology and the local data repository is more complicated. We use XPath expression templates (described later) for this purpose.

Query processing

When we want to access and manipulate the data in an XML document, we use the

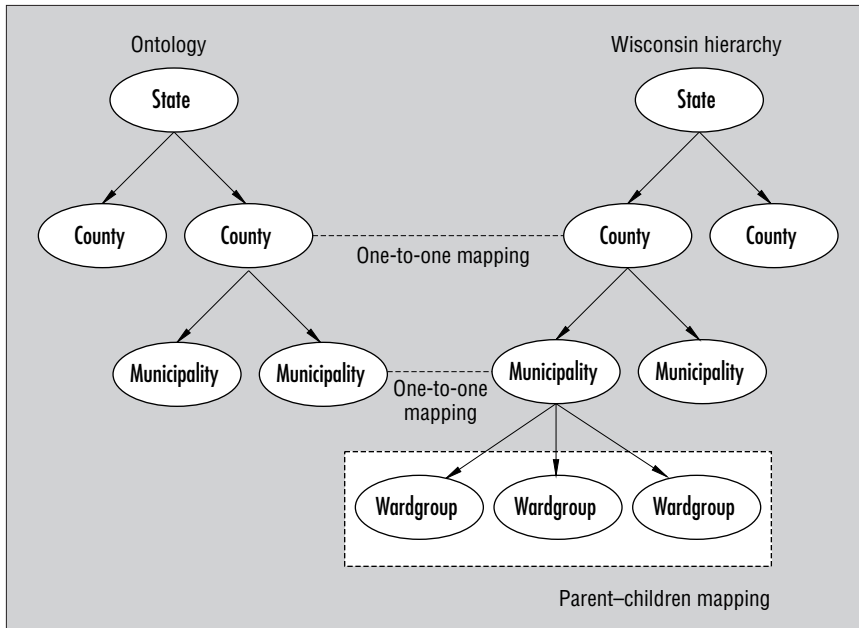


Figure 2. The parent-children mapping type.

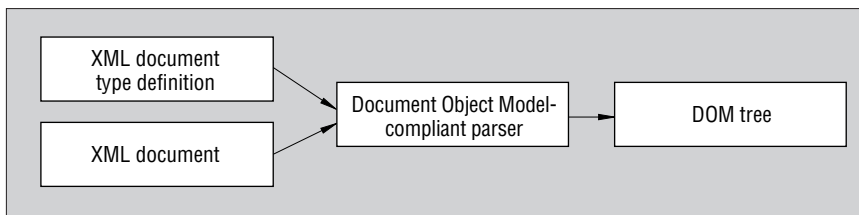


Figure 3. The parsing process.

```
<state name="Illinois">
  <county name="Adams">
  <county name="Cook">
</state>
```

Figure 4. XML data fragment used to illustrate XPath expressions.

Document Object Model. When we load an XML document using a DOM-compliant parser, the system returns a DOM tree (see Figure 3). You can create, access, or modify DOM trees within an application using functionality provided by open-source class libraries. A separate document, called the

Document Type Definition (DTD), specifies information about the hierarchy and the types of entities in the XML document.

A DOM tree represents the entire XML document. Each node in the tree represents a corresponding element in the document. For querying an XML document, we need a mechanism to select only the nodes of interest from the DOM tree. XPath is a W3C specification developed for this purpose. It provides a way to express a path through a document tree to select a set of nodes (see Figure 4).

XPath's addressing mechanism is similar to that used by file systems. For example, the XPath expression "state/county" selects the county nodes under a state. The XPath

expression `state[@name='Illinois']/county` selects the county nodes under Illinois—that is, the counties of Adams and Cook in the XML fragment in Figure 5.

The system expresses the user query in terms of the entities in the ontology. It then splits the user query into a subquery for each local data repository, which it expresses in terms of local entities. The system uses the agreement document information about the mappings between equivalent entities for the query-rewriting process.

Each entity in the ontology has a corresponding XPath expression template. The template encodes the mapping between the ontological entity and the equivalent entities in the local data repository. We classified these mappings into five different types and encoded each type using a different type of XPath expression template. An XPath expression template contains the XPath expression with placeholders for the arguments. The user supplies the arguments when submitting a query. The system substitutes these arguments in the template to obtain the appropriate XPath expression (see Figure 6).

The system then executes the XPath expression against the XML documents in the local data repository. The system returns the DOM nodes, which are equivalent to the ontological entities of interest specified in the user query. When the system executes an XPath expression, it returns the result in the form of textual data. Using information about the data type, the system can perform additional processing on this textual data. For example, in the case of integers, the user could perform integer-specific operations.

XPath expression templates and user interface

The Apache Xalan XPath API provides a set of classes for handling XML documents as DOM objects and for executing XPath expressions on those objects. We developed a framework of Java classes using the Xalan XPath API classes. The XPath expression templates can be either specified statically in the agreement document or constructed dynamically using knowledge about the different schemas and the entity of interest in the user query.

Executing an XPath expression involves traversing the levels in the local hierarchy. When only a fixed number of levels are to be traversed for finding the equivalent entities of any ontological entity, you can specify the XPath expression template statically.

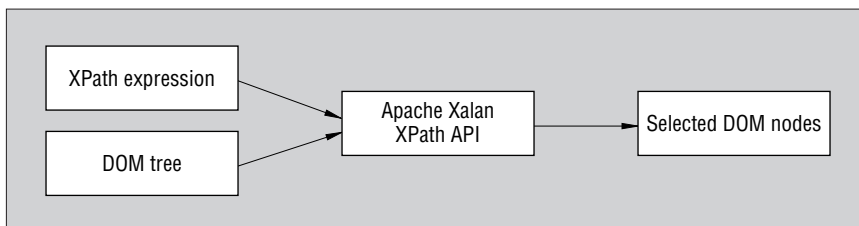


Figure 5. The querying process.

In our election query application, we specified the XPath expression templates statically in the agreement document. A state contains counties, each containing municipalities. This hierarchy results in a fixed number of levels to traverse for finding the equivalent entities. If the number of levels to be traversed is not fixed—as in our land-use-query application—the XPath expression template must be constructed dynamically when the query is submitted.

In either case, the system displays the results that the query-processing component collected from the local data repositories. Given the data's geographic nature and the need for its remote access, we designed the user interface using interactive maps in a Web browser. To implement the user interface, we used Axiomap⁹ for publishing interactive maps on the Web.

Application one: Election result queries

We designed our first application to help users browse the 2000 US presidential election results collected from different states and categorized according to various counties, municipalities, and other units. To analyze the election results at the national level, the system must retrieve data from different states and present it to the user in a common format. However, each state stores its election results using different levels of resolution. For example, Wisconsin stores its data for state, county, municipality, and ward group, while Illinois stores its data for only state and county. We downloaded results for the US presidential elections from the official Web sites maintained by the different agencies. Figure 7 shows a fragment of the Illinois election results.

The entities of interest in the ontology include *state*, *county*, *municipality*, and *candidate*. Each of these entities has a *name* attribute. The *candidate* entity has the additional attribute *votes*, which indicates the votes that each candidate secured. A state consists of several counties, each of which can have several municipalities. Each municipality has a set of candidates. The XML DTD in Figure 8 represents the ontology.

The local data repositories of all states have the *state* and *county* entities in their hierarchy. The differences arise in the organization of smaller geographic entities such as municipalities, precincts, ward groups, and wards. When the system calculates the total number of a candidate's votes, it specifies the XPath

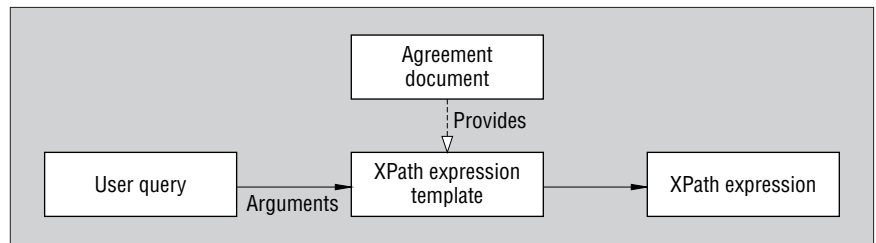


Figure 6. Substituting arguments in the XPath expression template.

```

<state name="Illinois">
  <county name="Adams">
    <candidate name="Gore" votes="12197"/>
    <candidate name="Bush" votes="17331"/>
    <candidate name="Nader" votes="371"/>
    <candidate name="Buchanan" votes="140"/>
    <candidate name="Browne" votes="63"/>
    <candidate name="Hagelin" votes="7"/>
    <candidate name="Phillips" votes="0"/>
    <candidate name="McReynolds" votes="0"/>
    <candidate name="Total" votes="30109"/>
  </county>
  ...
</state>
  
```

Figure 7. A fragment of the XML document containing the election results for Illinois.

<!ELEMENT state	(county+)
<!ELEMENT county	(municipality+)
<!ELEMENT municipality	(candidate+)
<!ELEMENT candidate	EMPTY
<!ATTLIST state	name CDATA #REQUIRED>
<!ATTLIST county	name CDATA #REQUIRED>
<!ATTLIST municipality	name CDATA #REQUIRED>
<!ATTLIST candidate	name CDATA #REQUIRED votes CDATA #REQUIRED>

Figure 8. The XML DTD representation of the election ontology.

```

<xpath-expr>
  /child::state[attribute::name='Wisconsin']\
    /child::county[attribute::name='$county_name']\
      /child::municipality[attribute::name='$municipality_name']\
        /child::*
</xpath-expr>
  
```

Figure 9. The XPath expression template to query all the candidates in a given municipality in a given county in Wisconsin.

expression template in the agreement document. As we mentioned before, this specification is static because it is the same for all counties. It depends only on the general relation between the *county* entities in both schemas and does not change with the spe-

cific value of the county of interest. The system passes the entities' names as arguments to the XPath expression template. For example, the system uses the XPath expression template of Figure 9 to query all the candidates within a given municipality in a given

```

<!-- select all candidate names, return a set of strings -->
<operation name='getCandidateNames' aggregation='sset'>
  <body>
    child::candidate/attribute::name
  </body>
</operation>

<!-- select total number of votes for a candidate, return an integer -->
<operation name='getVotesByCandidate' aggregation='isum'>
  <parameter>
    <name>$candidate_name</name>
  </parameter>

  <body>
    child::candidate[attribute::name='$candidate_name']\
      /attribute::votes
  </body>
</operation>

```

Figure 10. Aggregation operations.

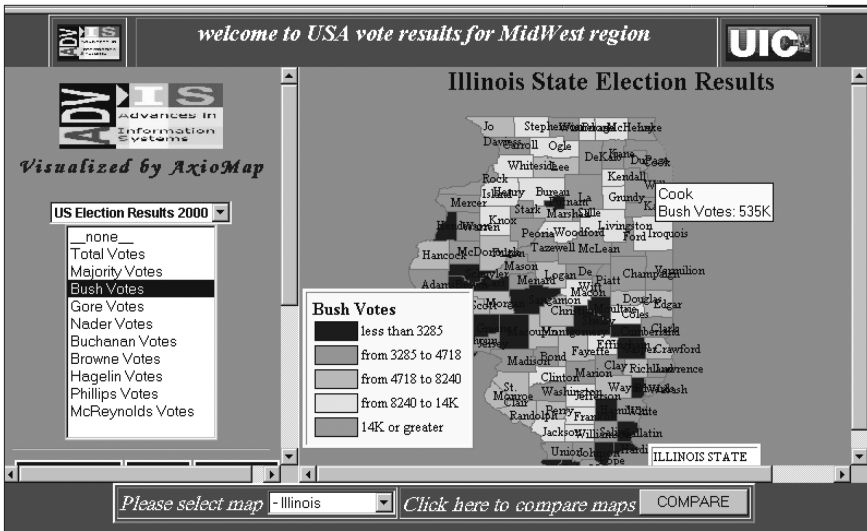


Figure 11. A map displaying election results for Illinois.

Table 1. Two land use classification schemes.

Exhaustive model		Hierarchical model	
009	Shopping center	1	Urban and developed land
010	Open water	1.01	Residential
111	Single family	1.01.01	Single-family detached or duplex
113	Two family	1.01.02	Mobile homes not in parks
116	Farm unit	1.01.03	Multifamily dwellings
140	Mobile home	1.01.03.01	Three-unit multifamily
		1.01.03.02	Four-unit multifamily

county in Wisconsin. The arguments to be passed from the user query are the names of the county (`county_name`) and municipality (`municipality_name`).

We identified some common user queries, including “List the candidates who competed in a given state or county or municipality” and “How many votes did a given candidate

get in a given state or county or municipality?” The system uses the `getCandidateNames` and `getVotesByCandidate` operations (see Figure 10) to answer these queries. Candidate names and votes are of types `string` and `integer`, respectively. Aggregation operations for strings and integers are different and are specified using the `sset` and `isum` operations.

As we mentioned earlier, interactive maps in a Web browser serve as the user interface. Initially, the system shows the user a page with selectable state areas. When the user selects a state, the browser directs the user to an analysis of its election results. In the map of Illinois in Figure 11, the system displays the number of votes that George W. Bush received relative to the total number of votes.

Application two: Land use queries

The *Wisconsin Land Information System* is a proposed distributed Web-based system with heterogeneous data on local and state servers. WLIS has to integrate data across jurisdictions by overcoming the syntactic and semantic heterogeneities that result from using different coding conventions in different data repositories. Table 1 shows two land use classification schemes.

A typical query—such as “Where are all the crop and pasture lands in Dane County?”—would be relatively straightforward when using one data set but more difficult when posed over a larger geographic area. Table 2 illustrates the heterogeneity of attribute names and values that would satisfy the criteria of the query over selected multiple data sets. `Lucode`, `Tag`, `Lu1`, and `Lu_4_4` must be resolved as synonyms for the attribute that represents the land use code in the ontology. Also, the Racine County data repository uses two land use codes for denoting croplands, while other counties use only one.

In land use data, the entities of interest are `land` and `owner`. Figure 12 shows sample XML data about a land parcel and its owner. Land parcel data for Dane County in Wisconsin is stored digitally and can be accessed on the Web (<http://wisclinc.state.wi.us/datadisc/orgmetabl1.html>). Our land parcel data contains an identification number for the parcel (`lid`), the category of land use under which it is classified (`lucode`), the file containing the pertinent shape information (`shape_file`), and information about the parcel’s owner (`owner_id`). Owner data usually contains the owner’s name (`name`), date of birth (`dob`), social security number (`ssn`), and gender (`gender`).

The election ontology's entities of interest include *database*, *table*, *tuple*, *attrname* (attribute name), and *attrvalue* (attribute value). Each of these entities has an *id* attribute. The *attrvalue* entity has the additional attribute *expansion*, which indicates the meaning associated with the predefined attribute value.

A database consists of tables, which consist of tuples. A tuple consists of a series of *attrname* entities, each of which might have predefined values. Our chosen organization of the entities in the ontology is similar to the frame-based notation that XOL uses.¹⁰ We represent the ontology as an XML DTD (see Figure 13). A typical land tax database system will have tables for *land* and for *person*.

As Figures 8 and 13 illustrate, the representation of the ontology for the land use application differs from that for the election results application. DTD elements that represent entities in the ontology for the presidential results correspond to the real-world entities of state and county in the problem domain. However, DTD elements for the land use application do not correspond to entities in the problem domain but to entities in a database schema, which we use to organize the problem domain. We made this decision to maintain the application's extensibility and to keep the hierarchy simple.

A simple land tax system contains information about land and people. But a more complicated system will have additional levels of information. Adding a separate entity, say *tax*, into an application in which the DTD represents database entities is simple. It simply involves adding an instance of *table*. If the DTD represents real-world entities, we will have to change the DTD each time we want to add a new entity.

In the *land_use_information* table, the *land_use_code* attribute has a set of predefined values.

```
<farm>
  <lid> lid21 </lid>
  <lucode> OA </lucode>
  <shape_file> 100 </shape_file>
  <owner_id> 124-45-5678 </owner_id>
</farm>
<owner>
  <name> John Doe </name>
  <dob> 12-30-1977 </dob>
  <ssn> 124-45-5678 </ssn>
  <gender> M </gender>
</owner>
```

Figure 12. Sample land use data.

Table 2. Heterogeneity of attribute names and values.

Planning authority	Attribute	Land use code	Description
Dane County Regional Planning Commission	Lucode	91	Cropland pasture
Racine County (Southeastern Wisconsin Regional Planning Commission)	Tag	811 815	Cropland pasture Other agriculture
Eau Claire County	Lu1	AA	General agriculture
City of Madison	Lu_4_4	8110	Farms

Representing each of these values as a separate entity clutters up the DTD without adding meaningful information. To keep the hierarchy simple, we represented the ontological land use codes in a separate document (see Figure 14). We used this document to resolve the differences in the land use coding conventions.

Each data repository might have a different organization of the attribute values depending on the primary function of the agency maintaining it. For example, a county where agriculture is the main occupation will have more categories of agricultural land than are in the state's ontology. When we map entities in the local data repository to those in the ontology, we might sacrifice data resolution to ensure conformance with a more general concept in the ontology. Our approach includes a facility for storing the different types of classification in the ontology to let us map local data repositories without losing data resolution.

For example, based on its function, we can classify commercial land use as *commercial sales* and *commercial service* in one local data repository and as *commercial intensive* and *commercial non-intensive* in another data repository. If the ontology supports only one type of classification, data in the other data repository must be aggregated into a single category. Our

approach makes it possible to preserve the resolution of data available in the local data repositories.

The local data repositories of all states have tables corresponding to *land* and *person* entities in their hierarchy. The differences arise in the syntax and the land use coding conventions. For example, in Dane County's hierarchy, the land use codes *35* and *36* represent *Scientific instruments* and *Miscellaneous industrial*, respectively. But the ontology does not have a separate land use code for *Scientific instruments*, so our system maps the collection of *35* and *36* to the ontological entity *010*, which represents *Industrial - Others* (see Figure 15).

The simplest mapping between entities such as *database*, *table*, *tuple*, and *attribute* in the ontology and those in the local data repository is one-to-one. Each entity's local equivalent is specified in the *equiv* attribute of the tag representing the entity. For example, the equivalences of the *land_use_information* table to the *tbl_land* table and the *land_id* attribute name to the *lid* attribute name are specified in the agreement document as

```
<table id="land_use_information" mapping="one-to-one" equiv="tbl_land">
<attrname id="land_id" mapping="one-to-one" equiv="lid" />
```

```
<!ELEMENT database (table+)>
<!ELEMENT table (tuple+)>
<!ELEMENT tuple (attrname+)>
<!ELEMENT attrname (attrvalue*)>
<!ELEMENT attrvalue (attrvalue*)>

<!ATTLIST database id CDATA #REQUIRED>
<!ATTLIST table id CDATA #REQUIRED>
<!ATTLIST tuple id CDATA #REQUIRED>
<!ATTLIST attrname id CDATA #REQUIRED>
<!ATTLIST attrvalue id CDATA #REQUIRED
expansion CDATA #REQUIRED>
```

Figure 13. The XML DTD representation of the land use ontology.

```

<database id="ontology">
...
<table id="land_use_information">
<tuple id="land">
  <attrname id="land_id"/>
  <attrname id="land_use_code">
    <attrvalue id="OC" expansion="Commercial">
      <attrvalue id="OCS" expansion="Commercial - Scale">
        <attrvalue id="OCSI" expansion="Commercial - Scale - Intensive"/>
        <attrvalue id="OCSN" expansion="Commercial - Scale - Non-intensive"/>
      </attrvalue>
      <attrvalue id="OCF" expansion="Commercial - Function">
        <attrvalue id="OCFL" expansion="Commercial - Function - Sales">
          <attrvalue id="OCFR" expansion="Commercial - Function - Service">
            </attrvalue>
          </attrvalue>
        </attrvalue>
        <attrvalue id="OCE" expansion="Commercial - Eateries">
          <attrvalue id="OCER" expansion="Commercial - Eateries - Restaurants">
            <attrvalue id="OCEB" expansion="Commercial - Eateries - Bars and Taverns">
              </attrvalue>
            </attrvalue>
          </attrvalue>
          <attrvalue id="OCO" expansion="Commercial - Others"/>
        </attrvalue>
      </attrvalue>
    </attrname>
    <attrname id="shape_file"/>
    <attrname id="owner_id"/>
  </tuple>
...
</table>
...
</database>

```

Figure 14. Predefined values of the land use code attribute.

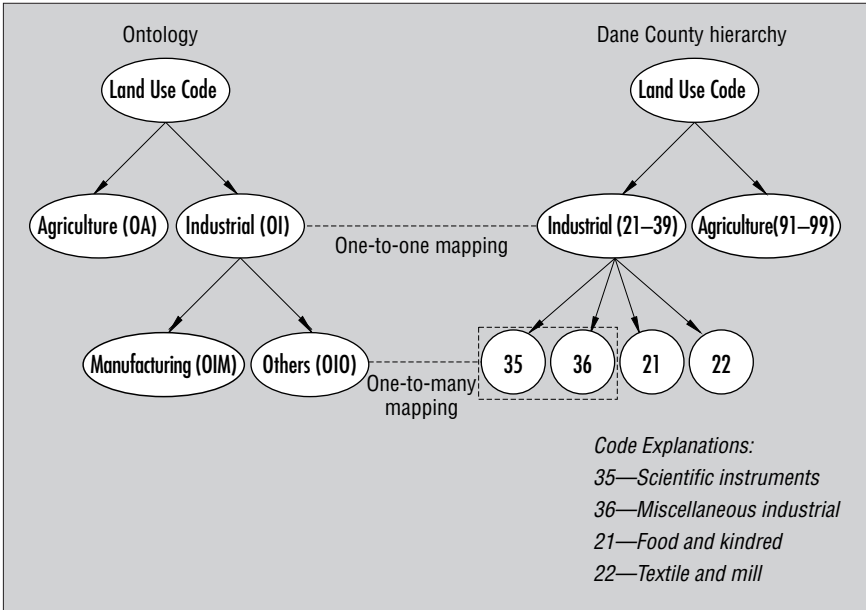


Figure 15. The one-to-many mapping type.

On the other hand, an ontology attribute value can map to a single value or a collection of attribute values in the local data repos-

itory. In some cases, multiple attribute values in the ontology can map to the same local attribute value. The specification of the rela-

tion between the entities in both schemas in terms of XPath expression templates was sufficient for querying the local data repositories in the election results application. But in this application, we had to construct the XPath expression templates on the fly according to information not specified in the agreement document.

The relation between entities such as **database**, **table**, **tuple**, and **attrname** is simple because they always occur at the same level in the classification scheme. But the relation between the **attrvalue** entity in both schemas is complex because the attribute values in a local data repository are grouped into different levels in the classification scheme. The number of levels to traverse for executing the user query on a land use code of interest depends on the level of nesting where that land use code occurs in the classification scheme. Therefore, we had to construct XPath expression templates dynamically.

This application's query interface differs from that of the election results application because there are no predefined queries. The user must be able to select some attributes from the tables in the database and specify constraining values for those attributes. Initially, the system shows the list of available local databases. The user selects the databases to be queried and then specifies the query in terms of the attribute names and values in the ontology. The system collects the results from all the selected databases and then displays them to the user in the form of interactive maps.

In Figure 16, the browser interface shows a small region of Dane county and the different land use codes. When the user selects a land use code in the left frame, the system highlights all land parcels in that region that fall under that classification. For example, in Figure 16 the user wants to find all parcels whose land use code is Agriculture: Cropland and Pasture. The system has highlighted Parcels P1 and P7, which fall under that classification. The user can learn more about a particular parcel by selecting it to open a pop-up window that displays additional information.

There are several areas for future work. XPath lets users traverse a linear path while collecting data from an XML document. But XPath does not support more complex mapping types, such as many-to-many. In addition, the data collected after the execution of an XPath expression is in the

form of text nodes. We had to provide an additional access layer to ensure type-safe access. Extensible stylesheet language transformations (XSLT) overcome some of these limitations by supporting many-to-many mappings and type-safe access.

To facilitate the creation of the agreement documents, we anticipate the need for a visual user interface for the local expert. With this interface, the expert would select an entity or a collection of entities in the ontology, the equivalent entity or collection of entities in the local data repository, and one of the four mapping options (one-to-one, one-to-many, many-to-one, or one-to-null). The expert would then ask the system to update the mappings until all the mappings are defined. This interface would hide the syntactic details, letting the expert concentrate on establishing the semantic connections.

To achieve complete generality and reap the full benefits of the emerging Semantic Web technologies, we intend to develop the connection between data and ontology at a higher layer of the interoperability hierarchy.¹¹ XPath expressions capture mapping semantics well, but an interesting possibility would be to capture the semantics using other kinds of expressions mapping ontologies represented using the resource description framework (RDF). ■

Acknowledgments

We thank Nancy Wiegand and Steve Ventura for discussions on land use problems, and Chaitan Baru and Ilya Zaslavsky for letting us use Axiomap. We also thank William Sunna, Paul Calnan, and Sathish Kumar Murugesan for help with the implementation. This research was supported partly by the National Science Foundation under Digital Government Award EIA-0091489, and by the Advanced Research and Development Activity and the National Imagery and Mapping Agency under award number NMA201-01-1-2001. The views and conclusions in this article are the authors' and do not necessarily represent the official policies or endorsements, either expressed or implied, of the National Imagery and Mapping Agency or the US Government.

References

1. I.F. Cruz et al., *The Emerging Semantic Web*, IOS Press, 2002.
2. Y. Bishr, "Overcoming the Semantic and Other Barriers to GIS Interoperability," *Int'l J. Geographical Information Science*, vol. 12, no. 4, 1998, pp. 299-314.
3. F. Fonseca and M. Egenhofer, "Ontology-Driven Geographic Information Systems," *Proc.*

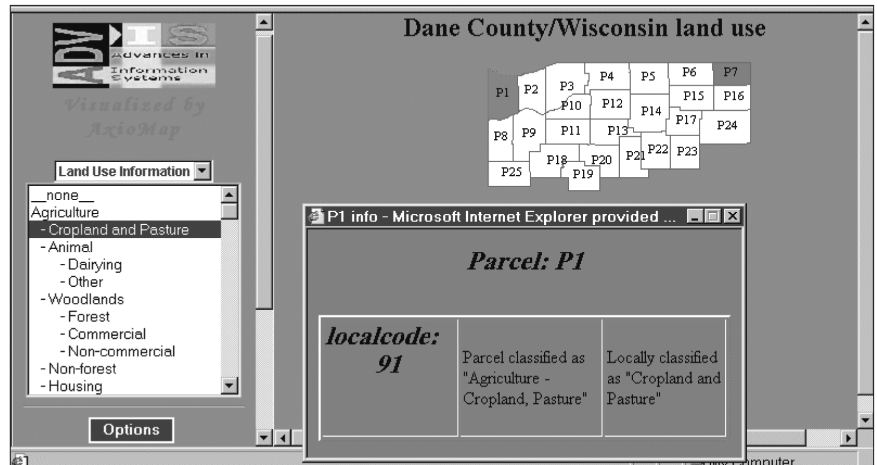


Figure 16. The query user interface, with interactive maps.

7th ACM Symp. Advances in Geographic Information Systems, ACM Press, 1999, pp. 14-19.

4. M. Gahegan, "Characterizing the Semantic Content of Geographic Data, Models and Systems," *Interoperating Geographic Information Systems*, M.F. Goodchild et al., eds., Kluwer Academic Publishers, 1999, pp. 71-84.
5. V. Kashyap and A. Sheth, "Semantic Heterogeneity in Global Information System: The Role of Metadata, Context and Ontologies," *Cooperative Information Systems: Current Trends and Directions*, M.P. Papazoglou and G. Schlageter, eds., Academic Press, pp. 139-178.
6. S. Abiteboul, D. Suciu, and P. Buneman, *Data on the Web: From Relations to Semistructured and XML*, Morgan Kaufmann, 1999.
7. V. Kashyap and A. Sheth, "So Far (Schematically) yet So Near (Semantically)," *IFIP Transaction A-25: Proc. IFIP WG 2.6 Database Semantics Conf. Interoperable Database Systems* (DS-5), North-Holland, 1993, pp. 283-312.
8. C.A. Cali et al., "Data Integration under Integrity Constraints," *Proc. 14th Int'l Conf. CAiSE*, LNCS, vol. 2348, Springer Verlag, 2002, pp. 262-279.
9. *Axiomap: Application of XML for Interactive Online Mapping*, Elza Research, 2001, www.elzaresearch.com/landv/axiomap_ie5.html.
10. R.E. Kent, "XOL," 1999, www.ontologos.org/Ontology/XOL.htm.
11. S. Melnik and S. Decker, "A Layered Approach to Information Modeling and Interoperability on the Web," *Proc. ECDL 2000 Workshop Semantic Web*, 2000, www.ics.forth.gr/isl/SemWeb/proceedings/session1-2/paper.pdf.

For more information on this or any other computing topic, please visit our Digital Library at <http://computer.org/publications/dlib>.

The Authors



Isabel F. Cruz is an associate professor in the Department of Computer Science at the University of Illinois at Chicago. Her research interests include databases and information systems, data integration, and the Semantic Web. She received a PhD in computer science from the University of Toronto. Contact her at the Department of Computer Science, Univ. of Illinois at Chicago, 851 S. Morgan St (M/C 152), Chicago, IL 60607-7053; ifc@cs.uic.edu; www.cs.uic.edu/~ifc.



Afsheen Rajendran is a graduate student in the Department of Computer Science at the University of Illinois at Chicago. His research interests include data integration for geographic information systems. He received a BEng in electrical and electronics engineering from the Birla Institute of Technology & Science. Contact him at the Department of Computer Science, Univ. of Illinois at Chicago, 851 S. Morgan St. (M/C 152), Chicago, IL 60607-7053; arajendr@cs.uic.edu; <http://cs.uic.edu/~advis/afsheen>.