Discovery of Schema Information from a Forest of Selectively Labeled Ordered Trees *

Dong-Yal Seo, Dong-Ha Lee, Kyung-Mee Lee, and Jeon-Young Lee

Dept. of Computer Science and Engineering
Pohang University of Science and Technology
Nam-Gu, Pohang, Kyongbuk, 790-784, KOREA
{dyseo, dongha, kyungmee, jeon}@white.postech.ac.kr

Abstract

The main focus of our work is to discover an object-oriented schema information from a set of semistructured data. We develop schema extraction algorithms and a data model for semistructured data . Our data model is an improved version of the data forest model. We modify the ordered labeled trees of the data forest model to allow selectively unlabeled vertices. The unlabeled vertices represent set structures which are syntactically incomplete in the data forest model. The efforts on schema discovery give a major distinction between our study and former ones. Schema extraction algorithms discover structural schema information from a set of semistructured data represented in our modified data forest model. We can extract identifiable classes, their attributes, and a composition hierarchy with the proposed algorithms.

1. Introduction

In conventional database systems, data objects are created as instances of predefined schema which reflects the structure of the real-world. But we should consider semistructured data in networked information world, where there is no absolute schema fixed in advance and whose structure may be irregular or incomplete. To manage such semistructured data efficiently, we need a new data model for the representation of schemaless data instances. And because such semistructured data have no absolute schema information and their structures are irregular, we should extract useful structural information and construct schema hierarchies to provide structured views and manipulation methods.

The WWW is a typical example of networked information environment. Many users creates their own documents and submit them through the WWW environment. We access various information resources with their URL addresses every day. But the structure of each HTML document is so semistructured, or almost unstructured, it is not easy to manage lots of information acquired from the WWW. If we could represent the information embedded in HTML documents and could define a well-structured schema on the set of documents, not only we have a better understanding of the collected information but also we can apply conventional database technologies which are mainly dependent on schema.

The importance and the motivation of semistructured data processing are introduced in earlier studies [1,2,3,4,5,6]. General introduction and survey on semistructured data processing are presented in [5]. All the earlier models for semistructured data [2,4,6] are similar to each other and their expressive powers are almost same. Earlier studies are focused on representation models and query languages. Although not developed as a semistructured data model, O_2's complex value model [7] provides a good example of syntactic representation and type system.

The OEM model[4] tried to represent semistructured data with attribute-value pairs. The values in the OEM model include sets and nested substructures as well as atomic values like integers and strings. The labeled-tree model[2,3], has the same expressive power as the OEM, represents semistructured data as trees, i.e., the trees with a labeling of edges. Our model is similar to the *data forest model* introduced in [6] by Abiteboul, et al. The data forest model supports *list* type which is unable to be described in the OEM and the labeled-tree.

The former studies mainly focus on modeling and querying with incomplete schema information or without it.

* This paper was supported by '95 SPECIAL FUND for UNIVERSITY RESEARCH INSTITUTE, Korea Research Foundation.

Although semistructured data have irregular structures and schemaless manipulation is convenient for users, schema itself is very important when we retrieve and manipulate the stored data. Schema provides a well-defined structural view of stored data, and enables us to use structured query languages, like SQL.
We are going to provide a way of schema-based management for semistructured data using object-oriented schema hierarchies. We suggest a schema extraction method and a suitable representation model of semistructured data to support schema-based manipulation. Our representation model is based on the data forest model.
The efforts on schema discovery give a distinction between our study and previous ones in semistructured data processing. Schema extraction will be much easier in a semistructured data set than from a set of wholly unstructured data because it has some structural information.
The remaining part of this paper is composed as follows. Section 2 introduces our model for semistructured data using the data forest model with selective labeling. Section 3 addresses schema extraction methodologies from a set of semistructured data represented in our data model. And finally, conclusion and directions for future work are discussed in Section 4.

2. The Data Model

We propose a new model for semistructured data, based on the data forest model. A data forest is composed of ordered labeled trees. An ordered labeled tree is a tree with a labeling of vertices and for each vertex, an ordering of its children. By the ordering property, the model supports *lists* which are not supported in the OEM and the labeled-tree. We modified the syntactic notation of ordered labeled tree to make it convenient to extract an object-oriented schema. We call the syntactically modified ordered labeled tree as *selectively labeled ordered tree(SLOT).*

2.1 Modified Data Forest Model with Selective Labeling

We define a modified version of the data forest model using the terms used in the original model. In our modified data forest model, we also assume the existence of some infinite sets introduced to define the data forest model: i) **name** of names; ii) **vertex** of vertices; iii) **dom** of data values. And we assume an additional finite set iv) **e-name** of a null string.
A modified data forest is composed of SLOTs. SLOT is an ordered labeled tree which allows an empty labeling of inner vertices. Without labeling, it represents an instance of its parent. And the parent vertex represents a set of unlabeled vertices. Empty labeling is not allowed for root and terminal vertices.
The internal vertices of the SLOTs have labels from **name** ∪ **e-name**, whereas the internal vertices of the ordered labeled trees have labels from **name** in the data forest model. All other properties of the SLOTs obey the constraints of ordered labeled trees. The semantic expressive power of SLOT is the same as that of ordered labeled tree. But SLOT distinguishes a singleton data element and a collection of elements syntactically.

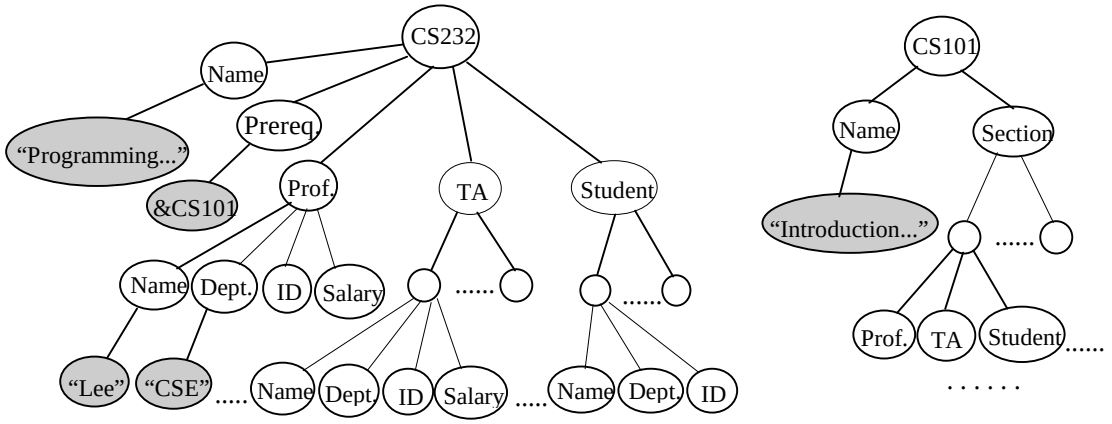

Figure 1 SLOT Representation Examples

Definition 1. A *modified data forest F* is a triple *(V,E,L),* where *(V,E)* is a finite ordered forest (the left-to-right ordering is implicit); *V* is the set of vertices; *E* the set of edges; *L* (the labeling function) maps some leaves in *V* to $V \cup$ **dom**; some internal vertex to **e-name**; and all other vertices to **name**.

Figure 1 shows SLOT representation of two courses in computer science. *CS101* is "Introduction to Computer Systems" and *CS232* is "Programming Language Exercise". *CS101* is a prerequisite for *CS232*. There can be more than one TAs and many students in a course. So the *TA* and *Student* vertices should be described as *sets* or *list*. Some courses are composed of two or more sections.

2.2 Management of Schema

If we managed the data set of Figure 1 in an object-oriented database, we may have an object-oriented schema like Figure 2. (The schema graph in Figure 2 is from [10].) There are 6 classes and corresponding relationships among the classes. The arrow lines show a class composition hierarchy and the thick gray arrows show a class inheritance hierarchy. For an example in class composition hierarchy, the class *Course* is composed of *Section* and itself. The class *Section* is composed of *Professor*, *TA*, and *Student*. The SLOTs in Figure 1 show the same kind of composition hierarchy. But the hierarchy in Figure 1 just reflects the structure of a single semistructured data item, not a common structure of a set of data items. For example, the two SLOTs in Figure 1 have different structures although they represent the same kind of data.

There are two important class hierarchies in an object-oriented database, the composition hierarchy and the inheritance hierarchy. The two hierarchies represent *IS-PART-OF* and *IS-A* relationships among classes respectively. In Figure 2, all the lines except for gray arrows can be simplified as *IS-PART-OF* relationships in an object-oriented database.

We are going to propose a schema extraction methodology to discover schema information as shown in Figure 2 from a set of semistructured data in Figure 1. Because a SLOT already shows an object composition structure, we may find a class composition hierarchy from a forest of SLOTs. Object composition structures are embedded in all other semistructured data models, not only in our SLOT representations.

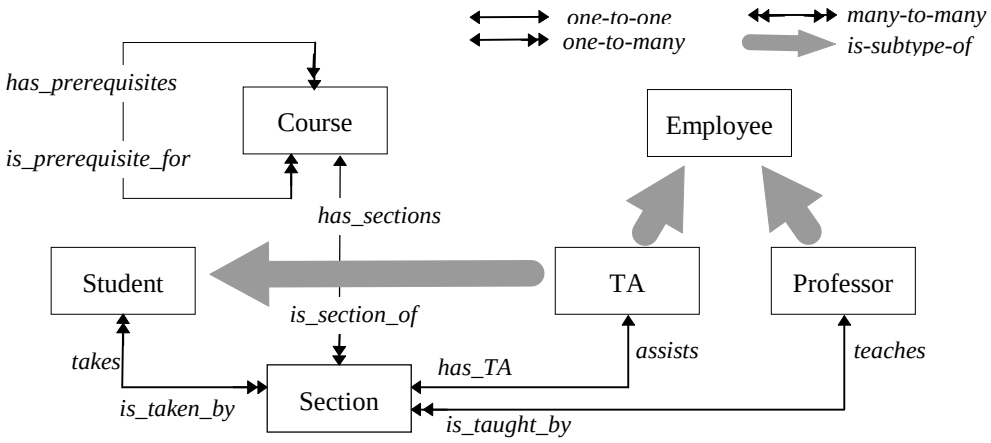

Figure 2 Class Hierarchies for Figure 1

3. Extraction of Schema Information

3.1 Class Composition Hierarchy

Because SLOT representations provide some structural information of data objects, the extraction of composition hierarchy and class structure can be performed by examining the forest of SLOTs[1].

At first we should identify possible vertices which will contribute as classes in an object-oriented database. Then we will find the attributes of the identified classes. Finally a possible class composition hierarchy will

[1] We assume that a forest of SLOTs represent a set of the same sort of data, e.g., course descriptions like Figure 1.

be discovered.

Step 1. Class Identification
The grand parents of the leaves represent identifiable classes or types. And the parents of leaves will contribute as the attributes of the grand parents of leaves. For example, the vertex *CS232* will be identified as a class which has attributes "*Name*" and "*Prereq.*". To decide identifiable classes and their attributes, eliminate the leaves and make their parents be the attributes of a class. *CS232* is just an instance of a class and not a class. The vertex *CS232* will be promoted as a class when we consider other object hierarchies and integrate them in further steps.

Step 2. Attribute Finding
The unlabeled vertices are the instances of their parent if we accept the parent as a class. Because there is no definition about any class, we should decide the structure of a class by examining the set of possible instances. The structure of a class must include the structures of all the children. To get a universally applicable structure of a class, eliminate unlabeled vertices and merge their children using the algorithm ***U-Merge***.

U-Merge(Vertex V)
for each unlabeled child U_i of vertex V
  if there exists a right-most sibling U_{i+1} of U_i
    merge U_i and U_{i+1} using the **Merge Rule**
  else
    eliminate U_i and make the children of U_i the children of V, the parent of U_i

Merge Rule If U_0, U_1 are unlabeled vertices to be merged and C_0, C_1 are the sets of their children, then the merged unlabeled vertex U_m has its set of children C_m where $C_m = C_0 \cup C_1$.

Step 3. Class Composition Hierarchy Construction
Applying Step 1 and 2 to the SLOTs in Figure 1, we have the composition hierarchies for individual data objects like the graphs in the left side of Figure 3. Now we should examine each object hierarchies and get the total class composition hierarchy from a set of those individual hierarchies. The class composition hierarchy should include all the object composition hierarchies. We can get a universal composition hierarchy by merging individual object hierarchies one by one.
Before we develop an algorithm to merge object hierarchies, we introduce the *super-concept* to decide merge condition.

Definition 2. Each vertex v has its n^{th} super-concept vertex $sup^k(v)$, e.g., $sup^1(CS232)=Course$. $sup^1()$ is specially called *immediate super-concept.* And we define a function of two vertices $lcsup(v_0,v_1)$ to get the lowest common super-concept of the two vertices. For example., $lcsup(CS232, CS101)=Course$. If $lcsup(v_0,v_1)=NULL$, it means that the two vertices are totally different concept and cannot be merged. The functions $lcsup()$ and $sup()$ have the following properties:

- $lcsup(v_0,v_1)=v \wedge \exists v'(sup^i(v_0)=v' \wedge sup^j(v_1)=v') \rightarrow \exists k(sup^k(v)=v')$
- $lcsup(v, sup^k(v)) = sup^k(v)$ *for all* $k > 0$.

We can manage the terms and functions for the *super-concept* using a *concept hierarchy*[8]. A concept hierarchy provides partially ordered relationships among concept terms according to a general-to-specific ordering. For example, in order to generalize *CS101* and *CS232* as *Course*, we need a concept hierarchy which contains the information like "$\{CS101, CS232\} \subset Course$". We can determine that *Course* is a common super-concept of *CS101* and *CS232* easily. But to find the lowest super-concept, we should examine all the relationships which contain *CS101* and *CS232*.
A concept hierarchy may be provided by users or domain-specific experts. Or we may use any kind of domain knowledge which represents the orders of relationships among concept terms. Some conceptual hierarchies can also be discovered automatically using some machine learning techniques[9].

Now we will get a merged hierarchy h_m from two object hierarchies h_0 and h_1 where h_0:(V_0, E_0, R_0) , h_1:(V_1, E_1, R_1), and h_m:(V_m, E_m, R_m) using the following ***H-Merge*** algorithm. *V*, *E*, and *R* denote vertices, edges, and root, respectively. By applying the following ***H-Merge*** algorithm for all the object composition hierarchies until there left only one hierarchy, we will get a large hierarchy graph which includes all the composition structure of individual data items. The hierarchy in the right side of Figure 3 is constructed from the hierarchies in the left side.

Hierarchy **H-Merge** (Hierarchy h_0, h_1)
/* ***N*** is 0 or 1, and ***$\bar{N}$*** is the complement of ***N***. The function *subh()* returns a sub-hierarchy of the given hierarchy
rooted at the specified vertex. */
if h_N is an empty hierarchy
return $h_{\bar{N}}$
else if *lcsup*(R_N, $R_{\bar{N}}$) ≠ NULL
R_m ← *lcsup*(R_N, $R_{\bar{N}}$)
for each child v_i of R_N
for each child v_j of $R_{\bar{N}}$
children of R_m ← *H-Merge*(*subh*(h_N, v_i),*subh*($h_{\bar{N}}$, v_j))
else if there exists any *R* s.t. *lcsup*(R_m,*R*)≠NULL and *R* is in the forest of ($h_{\bar{N}}$ - $R_{\bar{N}}$)
R_m ← $R_{\bar{N}}$
R in $h_{\bar{N}}$ ← *lcsup*(R_m,*R*)
for each child v_i of R_N
for each child v_j of $R_{\bar{N}}$
children of *lcsup*(R_m,*R*) in $h_{\bar{N}}$ ← *H-Merge*(*subh*(h_N, v_i),*subh*($h_{\bar{N}}$, v_j))
else
R_m ← new vertex
children of R_m ← R_N, $R_{\bar{N}}$
return the new hierarchy rooted at R_m

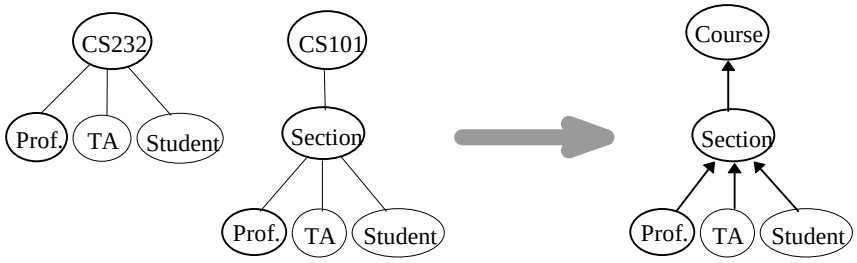

Figure 3 Possible Composition Hierarchy

The algorithm terminates when either of the input hierarchies is an empty one. If the roots of the two input hierarchies share a common super-concept, the algorithm builds a new root vertex which represents the lowest common super-concept of the two roots. Then merge their children and corresponding subhierarchies using the same algorithm recursively. For example, *CS232* and *CS101* in Figure 3 are generalized as *Course*, the lowest common super-concept.
When there is no common super-concept for two vertices, the two vertices are totally different concepts. But even the two vertices are totally different to each other, their subhierarchies may share a common concept. If *lcsup*(R_N, $R_{\bar{N}}$) = *NULL* but the root of one input hierarchy shares a common super-concept with the descents of the other one, the former root vertex is merged into the subhierarchy of the later one using the same algorithm recursively. For example, when we try to merge *Prof.* and *Section*, the children of *CS232* and *CS101* respectively, *Prof.* should be compared with the children of *Section* because it appears as a child of *Section*.

If $lcsup(R_N, R_N) = NULL$ and the two roots never share a common super-concept in the opposite-side sub-hierarchies, the two hierarchies are totally different concepts. The algorithm introduces a new vertex and attach the two input hierarchies as its substructures.

3.2 Problems and Discussions

Our algorithm is totally dependent on syntactic transformation of SLOT representations. But to construct a class inheritance hierarchy, we need more intelligent data processing like knowledge discovery and data mining[8] than simple graph restructuring methods. For example, if we want to find the gray arrows in Figure 3, we should be able to determine that *TA* is a specialized class of *Student*, and that *TA* and *Professor* can be generalized as the class *Employee*.

Cycle is a very sensitive property in an object-oriented class hierarchy. Our algorithm does not consider the cycling problem. For example, the cycle to/from the class *Course* in Figure 3 is omitted in Figure 3.

4. Conclusion and Future Work

The main focus of our work is to discover schema information for an object-oriented database from a set of semistructured data. We developed a schema extraction algorithm for semistructured data represented in our data model using SLOTs. Our model is based on the data forest model and modified to overcome its syntactic limitations. The algorithm will be refined and improved in the near future.

Integrating data mining techniques with semistructured data processing is a good issue as our next attack point. We expect that the solutions in that area will provide a nice clue for the construction of inheritance hierarchy. And it looks very useful when we want to refine the composition hierarchy discovered through the syntactic restructuring of SLOTs.

As an application example, we are developing a management system for the WWW resources. WWW is one of the largest data storage in the world but almost unstructured. To manage valuable information in the WWW more efficiently, we need a well-structured view of data and our schema extraction method will provide a way of management with object-oriented views.

References

[1] Dallan Quass, A. Rajaraman, Y. Sagiv, J. Ullman, and J. Widom, "Querying Semistructured Heterogeneous Information," *Proceedings of 4th International Conference on Deductive and Object-Oriented Databases*, Singapore, Dec. 1995, pp.319-344.

[2] Peter Buneman, Susan Davidson, Gerd Hillebrand, and Dan Suciu, "A Query Language and Optimization Techniques for Unstructured Data," *Proceedings of the '96 ACM SIGMOD*, Montreal, Canada, 1996, pp.505-516.

[3] Peter Buneman, Susan Davison, Mary Fernandez, and Dan Suciu, "Adding Structure to Unstructured Data," *Proceedings of the '97 ICDT*, Delphi, Greece, Jan. 1997, pp.336-350.

[4] Yannis Papakonstantinou, Hector Garcia-Molina, and Jennifer Widom, "Object Exchange Across Heterogeneous Information Sources," *Proceedings of the 11th IEEE International Conference on Data Engineering*, Taipei, Taiwan, March 1995, pp.251-260.

[5] Serge Abiteboul, "Querying Semi-Structured Data," *Proceedings of the '97 ICDT*, Delphi, Greece, Jan. 1997, pp.1-18.

[6] Serge Abiteboul, Sophie Cluet, and Tova Milo, "Correspondence and Translation for Heterogeneous Data," *Proceedings of the '97 ICDT*, Delphi, Greece, Jan. 1997, pp.352-363.

[7] Paris Kanellakis, Christophe Lecluse, and Philippe Richard, "Chapter 3: Introduction to the Data Model," *Building an Object-Oriented Database System: The Story of O_2*, Francois Bancilhon, Claude Delobel, and Paris Kanellakis, eds., Morgan Kaufmann, San Mateo, CA, 1992, pp.61-76.

[8] Ming-Syan Chen, Jiawei Han, and Philip S. Yu, "Data Mining: An Overview from Database Perspective," *IEEE Transactions on Knowledge and Data Engineering*, 1997.

[9] R.S. Michalski, "A Theory and Methodology of Inductive Learning," *Machine Learning: An Artificial Intelligence Approach*, vol.1, R.S. Michalski, G.C. Carbonell, and T.M. Mitchel, eds., Morgan Kaufmann, San Mateo, CA, 1983, pp.83-134.

[10] R.G.G. Cattell, ed., The Object Database Standard: ODMG-93 Release 1.2, Morgan Kaufmann Publishers, San Francisco, CA, 1996, pp.43-45.