

# Gmls : A Directed Graph Model with Label Sets for Semistructured Data Management

Ikumi Horie and Kazunori Yamaguchi  
The Graduate School of Arts and Sciences, The University of Tokyo  
{horie, yamaguch}@graco.c.u-tokyo.ac.jp

## Abstract

With the computerization of educational environments, the amount and types of student information available on computer have been increasing. However, it is difficult to handle such data in traditional databases, because student information has a lot of irregularity and representational diversity.

In this paper, we propose Gmls for managing data of such irregularity and diversity. First, we explain data structures and primitive operations. Then we show that applications can be built on Gmls easily. Finally, we show the result of handling sample data by our prototype Gmls system.

## 1. Introduction

With the computerization of educational environments, the amount and types of student information available on computer have been increasing. However, student information cannot be handled in the relational data model or object-oriented data models effectively because the student information has much irregularity and representational diversity.

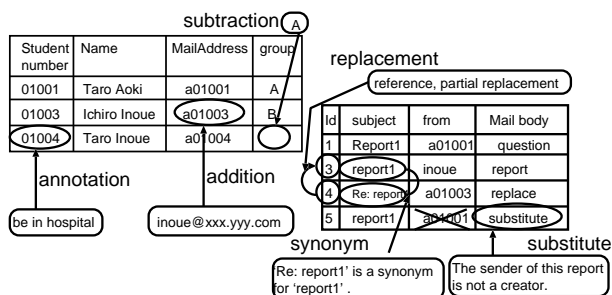


Figure 1. Examples of irregularity and diversity

Sample irregularity and representational diversity are listed below for the data in Figure 1.

- Irregularity

**annotation:** When a teacher has a student who is in hospital, he wants to put this fact on the student's record.

**addition:** Some student wants to use additional mail addresses.

**subtraction:** A student may leave a project group temporarily.

- Diversity

**replacement:** A student finds some mistakes in his submitted report and wants to submit another report to partially replace the previous one.

**substitute:** A student may ask his friend to send his report on behalf of him because he is ill.

**synonym:** A teacher wants to treat reports with the subject 'Re:report1' as those with the subject 'report1'.

These are just a few examples of irregularity and diversity. As these examples show, student information has a lot of irregularity and representational diversity. So, we need a data model for such data. In this paper, we propose Gmls for a model of the data.

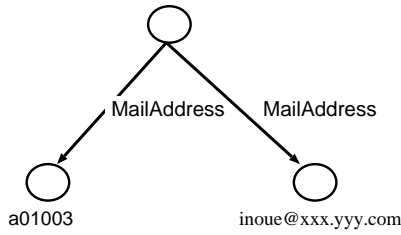
### 1.1. Related works

Many data models have been proposed so far. The relational data model [1, 2] and the object-oriented data models [3] are the most popular ones.

The relational data model manages a normalized relation, and schema is used to keep the validity of the database by forbidding the data which do not conform to the schema.

The object-oriented data models permit much richer structure than the relational data model does. However, it still requires that all data conform to a predefined schema. It is impossible to define a schema for unexpected irregularity and representational diversity. So these data models fall short of handling irregularity and diversity we are facing.

Lore[4, 5] is a database management system for semi-structured data, more specifically, XML. Lore consists of OEM, Lorel, and DataGuide. OEM is a graph, where nodes are objects and edges are labeled with attributes. OEM needs not conform to a particular schema. DataGuide is an application to extract useful information like schema from data. Lorel is a query language based on OQL extended



**Figure 2. An example representation in labeled graph**

with path expression, which is a pattern on a sequence of labels of target edges. For specifying a Lorel query, we have to know the schema at least roughly, even though we can use wild cards partially.

For managing educational information, there are many projects such as SCORM(Shareable Courseware Object Reference Model Initiative) [6]. However, the irregularity and representational diversity are not discussed intensively in these projects.

## 1.2. Outline of this paper

This paper is organized as follows. We introduce Gmls in Section 2. We explain data structure and basic operations. In Section 3 we show two sample applications of Gmls. The conclusion is found in Section 4.

## 2. Gmls

In this chapter, we introduce Gmls formally. Gmls stands for a directed Graph Model with Label Sets for Semi-structured Data Management.

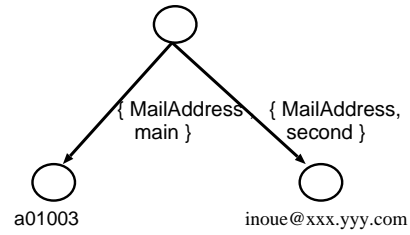
### 2.1. Data model

As is often adopted as an abstract model for semi-structured data, graph is adopted as an abstract model in Gmls, but differently from the previous models its nodes and arcs may have multiple labels in Gmls. In this paper, we call a set of labels *labelset*. For example,  $\{ 'a01003' \}$  and  $\{ 'MailAddress', 'main' \}$  are labelsets.

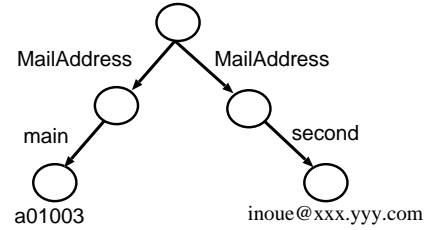
Formally, Gmls is defined as  $\langle N, A, L, C \rangle$  where  $N$  is a set of nodes,  $A$  is a set of arcs such that  $A \subseteq N \times N$ ,  $L$  is a set of labels, and  $C$  is a predicate on  $(N \cup A) \times L$ .  $C$  is meant to express that if  $C(x, l)$  holds, then the node or arc  $x$  has the label  $l$ . For a node or arc  $x$ ,  $\{ l \mid C(x, l) \}$  is called the *labelset on  $x$* .

Figure 2 shows an example representation of a conventional labeled graph in Gmls. As omitted in this figure, curly brackets and commas are omitted if there is no ambiguity.

If we want to distinguish two mail addresses, we can add the label *'main'* or *'second'* to each labelset as shown in Figure 3.



**Figure 3. Labelset example**



**Figure 4. Labelset in labeled graph**

Note here that there are two arcs with the label  $\{ 'MailAddress' \}$  in their labelsets. By following the arcs, we can get two nodes with the labelset  $\{ 'a01003' \}$  and the labelset  $\{ 'inoue@xxx.yyy.com' \}$ . If we include more labels in the labelset we can make a distinction between the arcs and the terminal nodes of them. For example, we can get one of the arcs by the labelset  $\{ 'MailAddress', 'second' \}$  whose terminal node is labeled  $\{ 'inoue@xxx.yyy.com' \}$ .

As this example shows, we can describe precise conditions as well as rough conditions on target nodes by specifying distinguishing or common labels.

In the conventional labeled graph, multiple labels have to be represented by the labels of the sequence of arcs as shown in Figure 4 or the labels of dangling arcs. In Gmls, multiple labels can be represented directly as shown in Figure 3.

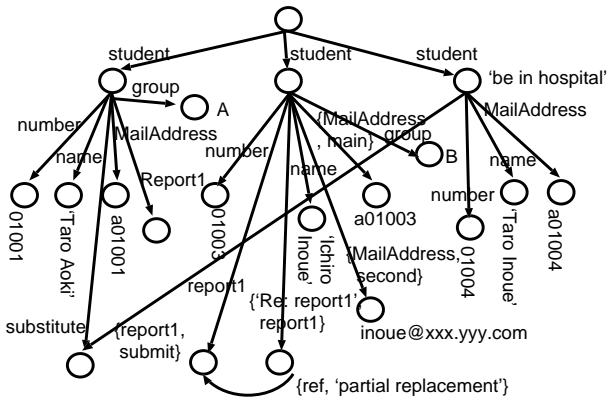
Now, we show how the sample data of irregularity and diversity in Figure 1 is encoded in Gmls in Figure 5.

**annotation:** The label *'be in hospital'* is added to the labelset of the node as an annotation.

**addition:** A new node for the additional mail address is created. Two nodes for the mail addresses are given the common label *'MailAddress'* and distinguishing label *'main'* or *'second'*.

**subtraction:** We can delete nodes and arcs.

**replacement:** New nodes for the succeeding reports are created and the arc from the new report to the previous



**Figure 5. The sample data of irregularity and diversity represented in Gmls**

one is created. When we grade reports we can see the report is partially replaced by following the incoming arcs.

**substitute:** A report and its sender are related by the arc with the label 'substitute', while a report and its writer are related by the label 'submit'. So, we can avoid the possibility that the report is mistaken to be submitted by the sender.

**synonym:** We add the label 'report1' to the labelset 'Re: report1'. Then, we can pick up nodes and arcs with the labelset 'report1' and 'Re: report1' by the common label 'report1' uniformly.

As these examples show, Gmls can represent data with irregularity and diversity of student information.

## 2.2. Primitive operation

Now we consider the primitive operations of Gmls.

We first review what kinds of operations may be needed for handling the sample data.

**annotation:** We want to find annotations even if we don't know whether annotations are attached to nodes or arcs. Here, we do not want to make a distinction between nodes and arcs.

**addition:** We need to send a mail to all the mail addresses of each student.

**subtraction:** We want to find the students who don't belong to any groups.

**replacement:** When multiple reports are submitted from a student, we want to know the relations among the reports.

**substitute:** We need to compare the report of the sender and the report of the creator.

**synonym:** When we find a divergent subject of reports such as 'report1' and 'Re: report1', we want to put the common label 'report1' to them so that they can be picked up by the common label 'report1'.

For constructing these operations, primitive operations should meet the following requirements.

- There may be many annotations, additions, and so forth. So nodes and arcs with the same labels should be able to be handled at a time.
- Target labels may be put on either nodes or arcs. So the primitive operations should make no distinction between nodes and arcs as much as possible.
- It must be easy to follow arcs.

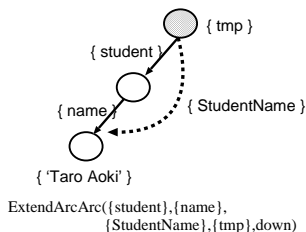
In order to satisfy these requirements, we introduce the following eight primitive operations.

Below, we say 'a node with labelset' to mean that the labelset of the node contains labelset and 'a node of labelset' to mean that the labelset of the node is labelset. We also say 'labelset is added to a node' to mean that labelset is added to the labelset of a node. We say 'labelset is subtracted from a node' to mean that labelset is subtracted from the labelset of a node. The similar statements are used for arcs. If the terminal node of an arc is the starting node of another arc, then we call the latter arc is the following arc of the former.

- Creation of nodes or arcs
  - *CreateNode(labelset)*  
A node of labelset is created.
  - *CreateArcs(labelset<sub>1</sub>, labelset<sub>2</sub>, labelset<sub>3</sub>)*  
Arcs of labelset<sub>3</sub> from nodes with labelset<sub>1</sub> to nodes with labelset<sub>2</sub> are created.
- Deletion of nodes and arcs
  - *Destroy(labelset)*  
Nodes and arcs with labelset are deleted.
- Addition of labels to and subtraction of labels from labelsets of nodes and arcs
  - *AddLabels(labelset<sub>1</sub>, labelset<sub>2</sub>)*  
labelset<sub>2</sub> is added to nodes and arcs with labelset<sub>1</sub>.
  - *SubtractLabels(labelset<sub>1</sub>, labelset<sub>2</sub>)*  
labelset<sub>2</sub> is subtracted from nodes and arcs with labelset<sub>1</sub>.

- Extraction of labelsets from nodes and arcs with  $labelset_1$ 
  - $GetLabels(labelset_1)$   
This operation returns all the labels of nodes and arcs with  $labelset_1$ .
- Traversing operation of nodes or arcs
  - $ExtendNodeArc(labelset_1, labelset_2, labelset_3, labelset_4, up | down)$   
If the fifth argument is ‘up’,  $labelset_3$  is added to the arcs with  $labelset_2$  whose terminal nodes have  $labelset_1$ , and  $labelset_4$  is added to the start nodes of the arcs.  
If ‘down’ is specified at the fifth argument, the starting nodes and terminal nodes are treated conversely.
  - $ExtendArcArc(labelset_1, labelset_2, labelset_3, labelset_4, up | down)$   
If the fifth argument is ‘down’, arcs of  $labelset_3$  are created from the starting nodes of arcs with  $labelset_1$  to the terminal nodes of the following arcs with  $labelset_2$ . When the arcs exist already,  $labelset_3$  is added to the arcs. Then,  $labelset_4$  is added to the starting nodes of the created arcs.  
If ‘up’ is specified at the fifth argument, the starting nodes and terminal nodes are treated conversely.

Figure 6 shows an example use of  $ExtendArcArc$ . In this figure, the dotted arc with ‘ $StudentName$ ’ is created as the shortcut of an arc with ‘ $student$ ’ and the following arc with ‘ $name$ ’. The label ‘ $tmp$ ’ is added to the starting node.



**Figure 6.** An example use of  $ExtendArcArc$

By these primitive operations, the irregularity and diversity of the sample data can be handled as follows.

**annotation:** By  $GetLabels$  for the labelset {‘ $be in hospital$ ’}, we can get all labels related to the annotation.

**addition:** By  $ExtendNodeArc$ , we can follow arcs labeled {‘ $MailAddress$ ’}.

**subtraction:** By  $AddLabels$ , we can add the label ‘ $NoGroup$ ’ to nodes. Then, by  $ExtendNodeArc$ , the label ‘ $mark$ ’ is added to the starting nodes of the arcs with ‘ $group$ ’. Then, by  $SubtractLabels$ , the labelset {‘ $NoGroup$ ’, ‘ $mark$ ’} is subtracted from the nodes with labelset {‘ $NoGroup$ ’, ‘ $mark$ ’}. The remaining nodes with ‘ $NoGroup$ ’ are the targets.

**replacement:** By  $ExtendNodeArc$ , we can add a specific label to the arcs between the given node of ‘ $report$ ’ and the other nodes of ‘ $report$ ’. Then by  $GetLabels$  for the specific label we can get other labels of the arcs, which are useful for understanding the relationship between the reports.

**substitute:** By repeatedly applying  $ExtendNodeArc$ , we can create an arc between a report with a student as a creator and the one with the student as a substitute. Then, we can check whether reports related by the arc are similar or not.

**synonym:** By  $AddLabels$ , we can add the common label ‘ $report1$ ’ to all the nodes labeled ‘ $Re: report1$ ’. Then, we can extract all the labels of report1’s by  $GetLabels$ .

### 3. Application

In the previous section, we introduced very primitive operations of Gmls. For handling the semantics of student information, complex applications are needed. In this section, we show that such complex applications can be developed using the eight primitive operations effectively. As sample complex applications, we introduce a table form viewer and concept analyzer.

#### 3.1. Table form viewer

A part of data in Gmls might have regular structure such as table.

Suppose that names, addresses, regular reports, programs, and mails are stored in Gmls, and we need the list of students and their reports. In this case, we want to view a name and report as a record in table form. For this purpose, we introduce a table form viewer.

This table form viewer shows a table which is encoded in Gmls under an encoding rule specific to the viewer. The sample data encoded in Gmls and its table form are shown in Figure 7. In this example, the table is supposed to have ‘ $MailAddress$ ’, ‘ $name$ ’, and ‘ $number$ ’ as attributes.

Note that a table handled here is slightly more general than the normalized relation in that the table may have a repeating group in its fields.

Figure 8 shows that three tables are generated from a single graph.

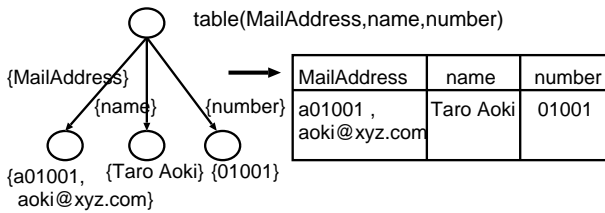


Figure 7. Table view

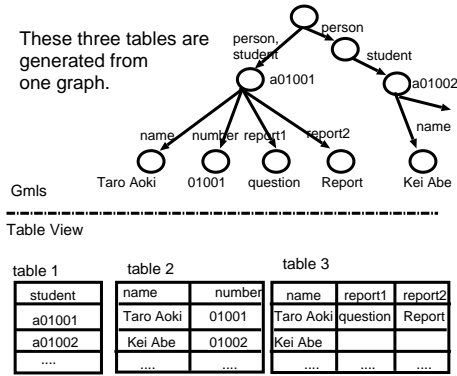


Figure 8. Three table views for different attributes

Even though the encoding is fixed, we can modify a graph in Gmls to fit to the encoding so that it can be visualized in a table form. Figure 9 shows the example of a table view for a modified graph. By the primitive operations, a node is created for each group and an arc is created from a group to each member of the group. Then we can visualize group and its student list in a table form.

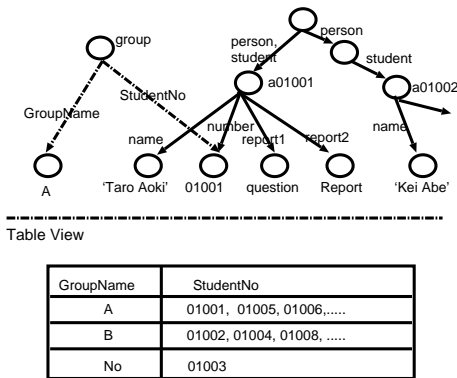


Figure 9. Table view for modified graph

### 3.2. Concept analyzer

Suppose that students have submitted multiple reports. In this case, for proper handling of reports a teacher needs to know the relationship between the reports. For this purpose, the teacher has to know the right labels such as 'replace' or 'follow-up' to pick up the arcs between the reports. In order to find out the relationship between labels, we may employ concept analysis by Galois lattice[7] by regarding nodes and labels as objects and attributes, respectively.

Figure 10 is an example of concept analysis. A Galois lattice is generated for the binary relation between nodes of three reports and labels 'mail', 'mule', 'Internet', 'usage', 'editor', and 'html'.

From this lattice, we see that reports #1 and #2 are similar, 'mail' and 'mule', or 'mail', 'Internet', and 'html' are in groups, while 'mail', 'mule', and 'Internet' are not in groups.

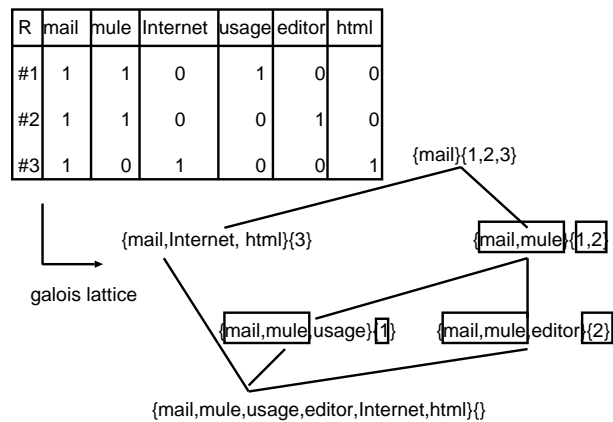


Figure 10. A binary relation and Galois lattice

### 3.3. Prototype system

We have implemented a prototype system of Gmls in Java, and used reports of an introductory Java class as sample data of student information in the prototype Gmls system.

Figure 11 shows a part of the sample data and Figure 12 shows the table form viewer. Figure 13 shows the preliminary concept analyzer.

### 4. Conclusion

The experiment by the prototype Gmls system for the actual data showed that Gmls is promising.

We have implemented just two preliminary applications. For practical use, more and more applications should be developed, and their mutual interference on Gmls has to be carefully evaluated.

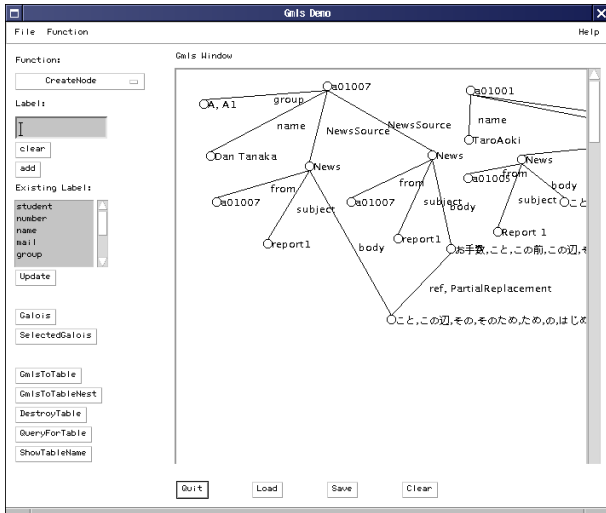


Figure 11. Student information in the prototype system

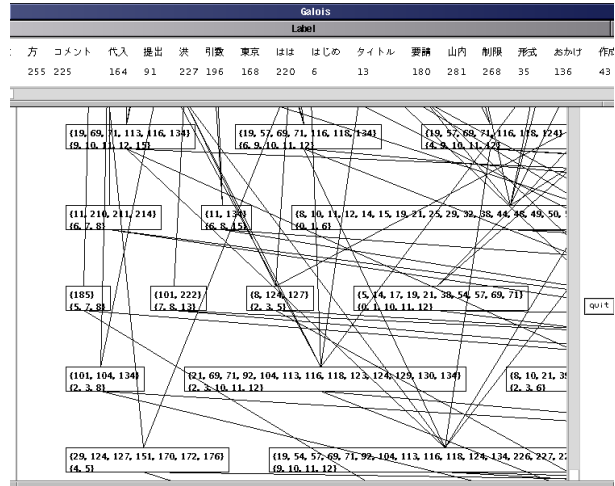


Figure 13. Concept analyzer in the prototype system

There may be relations between labels, and some explicit support for the relations may be useful. We want to extend Gmls in this direction.

We also want to apply our prototype system to more actual data.

## 5. Acknowledgment

This work was partially supported by Grant-in-Aid for Scientific Research(A) "Advanced Utilization of Multimedia to Promote Higher Educational Reform" of Ministry of Education, Culture, Sports, Science and Technology of Japan.

## References

- [1] Jeffrey D. Ullman, "PRINCIPLES OF DATABASE AND KNOWLEDGE - BASE SYSTEMS - VOLUME I," Computer science press, 1988.
- [2] Jeffrey D. Ullman, "PRINCIPLES OF DATABASE AND KNOWLEDGE - BASE SYSTEMS - VOLUME II," Computer Science Press, 1988.
- [3] Alfons Kemper, Guido Moerkotte, "Object-oriented database management," Prentice Hall, 1994.
- [4] Abiteboul, S., "Querying Semi-Structured Data," Proceedings of International Conference on Database Theory '97, 1997.
- [5] McHugh, J., Abiteboul, S., Goldman, R., Quass, D., Widom, J., "Lore: A Database Management System for Semi-structured Data," SIGMOD Record, 26(3), September 1997, pp.54-66.
- [6] SCORM Home page, <http://www.adlnet.org/Scorm/scorm.cfm>
- [7] Gregor Snelting., "Concept Analysis - A New Framework for Program Understanding," ACM SIGPLAN/SIGSOFT workshop on Program analysis for software tools and engineering , 1998, pp.1-10.

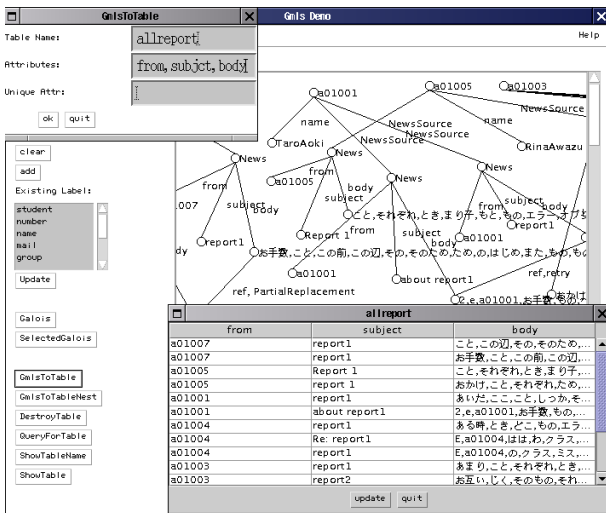


Figure 12. Table form viewer in the prototype system