Unifying Data Grouping and Knowledge Grouping through Nested Relation Based Knowledge Representation

Qiming Chen

Computer Science Dept.
University of California
Los Angeles, USA
qiming@cs.ucla.edu

Yahiko Kambayashi

Computer Science Dept.
University of Kyoto
Kyoto, Japan
yahiko@kuis.kyoto-u.ac.jp

Abstract

Providing a common scope for grouping subject-specific data and knowledge is the key requirement for database and knowledge base integration. The present object-oriented approach does not support such a requirement since data organization is class-based and knowledge organization is subject-based whose scopes do not coincide each other, and maintaining a large amount of actual classes for all the specific subjects is rather inefficient. In this paper an approach for unifying subject-specific data grouping and knowledge grouping through derived database patterns is proposed. As database patterns can be specified in terms of nested relations, a nested relation based knowledge representation framework is developed to support this approach.

Unlike previous approaches for complex objects reasoning which introduced ad-hoc Logic Programming (LP) notions but without extending underlying set manipulations, the proposed framework is characterized by extending the fundamental notions at set theory level to underlie the notions at LP theory level. As a result, it has strong expressive power for dealing with hierarchically nested tuples and sets, is canonical and analogous to first-order LP framework.

1. Introduction

Coupling data grouping and knowledge grouping is the key requirement for integrating database and knowledge base systems. Presently there exists a strong tendency of eliminating the border between data organization and knowledge organization by turning Object-Oriented Data Bases (OODB) to Object-Oriented Knowledge Bases (OOKB). However, to reach such a goal with satisfactory results is inherently difficult since data organization and knowledge organization have different scopes, natures and emphases as described in the following.

Knowledge organization vs. data organization

Data organization is data domain oriented. Grouping objects into a class is based on the common properties shared by all the objects of that class. For example, the class "PROFESSORS" encapsulates the properties commonly applicable to all the professors, while the specific knowledge applicable to only few individual professors is not considered as general properties of this class.

Knowledge organization is subject domain oriented. The objects involved in a subject form its object scope, and the rules about that subject are locally applicable to these objects. The semantic links between subjects actually describe the relationships between the objects within the object scopes of the subjects.

From data organization point of view, objects of multiple classes may be involved in one subject, and objects of a single class may be involved in multiple subjects. Further, an object may play different roles under different applications and therefore relating to different knowledge, such as a professor involved in a course as an instructor, in a project as a researcher, or in a company as a consultant.

From knowledge organization point of view, the object scope of a subject may involve objects from multiple classes, but a rule may be applicable to only few objects of a class containing thousands of objects. For example, the knowledge about project "IBM89" is applicable to only the professors and students participating that project, rather than the whole class of "PROFESSORS" or "STUDENTS"; the knowledge about "after-earthquake-first-aid" is exclusively applicable to the residents living in the quake area, rather than all the residents.
In summary, a class is usually too general to underlie the detailed semantics of specific subjects, and a subject is usually too comprehensive to couple object classes. Since a database should not be restricted to support a single application, inherently there exists a gap between general data modeling and specific data applications. Although OODB’s allow creating specific classes to coincide the scope of subject knowledge grouping, maintaining a large amount of actual classes for dealing with various specific cases would considerably increase the system overhead.

**Dynamic application vs. static classification**

Object classification is *static* and predefined. The state of an object may vary but its type (or types) is rather stable and independent of its states. The type of an object can be changed only by explicit "schema evolution operations".

However, the object scope of a subject is generally *dynamic* rather than *static*. For instance, the residents falling into the object scope of the subject "after-earthquake-first-aid" may not be identified and grouped in advance; A state alteration (e.g. change home address) may not automatically alter the type of an object, but may cause it falling inside or outside of the object scope of a particular subject.

In summary, it is difficult to specify all the possible applications in advance, and updating data and updating knowledge have side effects to each other which may force objects migration between classes.

To remedy the above problems we shall propose an approach for integrating class-based data grouping and subject-based knowledge grouping through database patterns derived from a database and specified in terms of nested relations, and develop a nested relation based knowledge representation formalism to underlie this approach.

2. **Pattern-based Data/Knowledge Grouping**

In order to integrate data/knowledge bases by reasonably coupling data organization and knowledge organization, we propose an Database - Pattern - Knowledge base framework (Figure 1) where data are grouped into classes and knowledge is grouped under subjects. A database and a knowledge-base are coupled via virtual patterns containing objects derived from the database and falling into the object scopes of specific subjects. Such a framework supports both *static organization* and *dynamic linkage* of data and knowledge.

A database pattern is defined on a single class or on multiple classes by specifying query conditions or deductive rules. The objects matching a pattern by satisfying the pattern condition form the *instance* of that pattern. A pattern instance contains the database objects involved in a particular subject, and the knowledge about that subject is localized to these objects. Therefore, this approach can be used to unify the object grouping and the knowledge grouping for a subject. It refines the object-oriented data/knowledge organization paradigm and supports dynamic applications. Since database patterns are virtual "views" of a database, generating them does not increase the database system burden for handling an extra amount of actual data classes, and modifying data and knowledge can be made independently without influencing each other. While the usual object-oriented approach organizes data and knowledge through *static classification*, the proposed approach does it through dynamic classification.

![Figure 1: An DB-Pattern-KB Architecture](image)
part instances of each relation. The knowledge grouped against this nested relation is exclusively applicable to it, rather than commonly applicable to all the instances of relation "PROFESSORS" and "STUDENTS". Moreover, such a nested relation is dynamically derived from the database and can be re-generated upon database or knowledge base updates.

![Knowledge Base Diagram](att1)

Figure 2: A Database Pattern in Nested Relation Form

3. Nested Relation Based Knowledge Representation

Since the proposed approach for integrating a database and a knowledge base is based on database patterns specified in terms of derived nested relations, it is necessary to develop a nested relation based knowledge representation formalism as the theoretical foundation of this approach.

3.1 Basic Requirements

Although reasoning with nested relations has received a great deal of attention over the past few years [APT86] [BEE86] [MAI86], the previous systems are insufficient for supporting the proposed data/knowledge-base integration approach since they do not meet the following key requirements:

1. strong expressive power for dealing with hierarchically nested tuples and sets,
2. canonical framework which is analogous to first-order LP framework.

We shall introduce a two-level view to the LP model theory: a higher level called LP theory level where the basic LP notions are described, and a lower level called set theory level where the fundamental set operations are provided to underlie LP notions. In first-order LP the basic notions at LP theory level are expressed in terms of single-level set manipulations at set theory level, and single-level set manipulations are incapable of dealing with structures. This can be considered as the reason for the mismatch between the conventional LP theory and complex objects. To deal with nested relations, it is necessary to extend the underlying set manipulation notions at set theory level to reformulate LP notions at LP theory level.

However, previous LP systems developed for dealing with complex objects did not extend single-level set manipulations. Instead, they introduced ad-hoc LP notions. As a result, they do not have full expressive power for handling mutually nested tuples and sets and for supporting nesting/unnesting, and do not have canonical frameworks since certain typical LP theorems, such as the uniqueness of least model, do not hold. While many present LP systems are claimed to be higher order they are not really based on high-order logic, in the sense that in general predicates (or types) used in them cannot be unknown variables. However, while those systems are essentially remained to be first-order, their semantic frameworks lack analogy with that of first-order LP. According to the above requirements they are insufficient for nested relation based knowledge representation.

In contradistinction to the previous LP approaches our approach is characterized by extending the fundamental notions at set theory level to underlie the notions at LP theory level. The framework built in this way is rather canonical in maintaining the major LP notions and rich in the expressive power required for nested relation based knowledge representation. We have used such a methodology in developing the nested relation oriented LP language HILOG-R [CHEN91].

3.2 The HILOG-R Language

HILOG-R is a LP language with high-order constructs used to handling nested relations. Symbols utilized in HILOG-R include constants, variables, types, attribute names, fillers, logical connectives and comparison operators. Two basic kinds of structure types: the tuple-type with a list of attribute types, and the set-type with a unique member type are introduced which may be mutually nested to form more complex types. To be clear
throughout this paper set-types are denoted by capital
symbols. The fixed-position constraint of attributes is re-
lixed, and specifying an attribute and a type by the same
name is allowed. For example, nested relation schemes
PROFS, STUDENTS and PROJECTS are specified by
the following types:

PROFS\{prof\{name:name, fields:SUBJECTS(subject),
  projects:CONTRACTS(contract)\}\}

STUDENTS\{student\{
  #\char, name:name, major:subject, project:contract\}\}

PROJECTS\{project\{
  title:contract,
  design_group:RESEARCHERS[
    researcher\{name:name, fields:SUBJECTS(subject)\}],
  implementation_group:PROGRAMMERS[
    programmer\{name:name, major:subject\}].
\}\}

An instance or a variable of a type is called a
term of that type. A nested relation is represented as a
nested term composed of simpler terms. For example, a
tuple instance of type project appearing in Figure 2 is ex-
pressed by the following term:

project\{title:IBM89,
  design_group:RESEARCHERS[
    researcher\{name:Smith, fields:SUBJECTS(AI),
        major:subject\}],
  implementation_group:PROGRAMMERS[
    programmer\{name:Joe, major:DB\},
    programmer\{name:Smith, major:KB\}].
\}\}

The special notation \texttt{R(x)}, where \texttt{x} is a variable
or a term containing variables, denotes a set-term of arbi-
trary cardinality. Strong typing and Unique Name As-
sumption (UNA) on types are two key syntactical con-
straints on terms. Formulas are defined as top-level struc-
tures, namely, top-level set-terms and tuple-terms, or a
comparison operator with arguments. Such a treatment
follows two considerations: first, to deal with a nested re-
lation as a hierarchical structured and integrated object;
second, to allow a relation (can be nested) which is a part
of one or more other nested relations, be handled individu-
ally. A conjunctive formula is in the form of \texttt{F\_1 \& F\_2}
where \texttt{F\_1} and \texttt{F\_2} are formulas. Then we can syntactically
define a HILOG-R rule as \texttt{head \leftarrow body}, where the \texttt{body}
is a formula and the \texttt{head} is a positive atomic formula, a
fact as a ground rule without a body, a HILOG-R pro-
gram as a finite set of rules, and a query as a rule without
a head. These notions provide us the means of manipulat-
ing nested relations, representing knowledge associated
with nested relations, and deriving new nested relations
from existing ones.

The semantic framework of this language is
developed through introducing at set theory level the no-
tions of partial inclusion and ungrouping.

The introduction of partial inclusion aims at han-
dling structural relationships between formulas and sets of
formulas. This notion comes from the typical part-term
relationship \leq between terms. Based on the part-term
relationship, the notions of partial-membership \in: and
partial-containment \subset: can be introduced, where the lat-
tice property of partial-containment allows us to further
induce the operations of partial-union \cup: and partial-
intersection \cap: We generally call these relationships as
partial inclusion. Below are some examples.

\texttt{SUBJECTS(DB, KB) \leq:
  researcher\{name:Parkcr, fields:SUBJECTS(DB, KB)\}).

\texttt{SUBJECTS(DB, KB) \in:
  \{researcher\{name:Parkcr, fields:SUBJECTS(DB, KB)\},
  researcher\{name:Smith, fields:SUBJECTS(AI, KB)\}\}).

\texttt{\{SUBJECTS(DB, KB) \subset:
  \{researcher\{name:Parkcr, fields:SUBJECTS(DB, KB)\},
  researcher\{name:Smith, fields:SUBJECTS(AI, KB)\}\}).

However, there exist other structural rela-
tionships between HILOG-R terms which cannot be directly
described by the partial inclusion notions. As it is unreas-
sonable to handle atoms or interpretations based on multi-
ple relationships, we shall introduce appropriate mappings
to convert atoms and sets of atoms into a form that can be
compared based on the unique partial-containment, such a
form is called the "ungrouped" form.

Ungrouping and grouping are used to handle
formulas containing set-terms. In general an atom with a
typed component set-term can be ungrouped into a set of
atoms with a single-element component set-term of the
same type; an atom with multiple set-terms is stepwise
ungrouped; a set of atoms is ungrouped into the union of sets
resulted from ungrouping each atom. Ungrouping a
single atom or a set of atoms always returns a set of
atoms, in this resulting set any set-term at any level of any
atom has one element. Such a treatment is iteratively ap-
plied to more complex formulas, for example,

\texttt{\{prof\{name:Smith, fields:SUBJECTS(AI, KB),
  projects:CONTRACTS(DB, KB)\},
  prof\{name:Smith, fields:SUBJECTS(AI),
  projects:CONTRACTS(DB, KB)\},
  prof\{name:Smith, fields:SUBJECTS(KB),
  projects:CONTRACTS(DB, KB)\},
  prof\{name:Smith, fields:SUBJECTS(KB),
  projects:CONTRACTS(IBM89)\},
  prof\{name:Smith, fields:SUBJECTS(KB),
  projects:CONTRACTS(IBM89)\}.}
Grouping is the transformation defined on a set of atoms for converting the set into the grouped form. Grouped formulas are preferred in representing nested relations. Although in certain cases grouping may not yield a unique result, in these cases the results represent the same semantics under UNA on types. Grouping is not simply the inverse of ungrouping. To group a set of atoms I, we first ungroup it to I', followed by grouping I'.

The grouping/ungrouping conversion is meaningful under UNA on types. Note that ungrouping is not used by users to model, describe or classify nested relations. It is not a syntactical construct of the language. It is only a tool used to assist expressing LPP semantics in dealing with logic satisfaction, model comparison, etc. When formulas (nested relations) or set of formulas can not be compared directly, we compare their ungrouped forms.

Partial-containment and ungrouping serve as the fundamental notions of HILOG-R model theory. For example, given an interpretation I

\begin{verbatim}
    \{ RESERCHERS |
        RESEARCHER(name:Smith, fields:SUBJECTS(AI, KB)),
        RESEARCHER(name:Parker, fields:SUBJECTS(DB, KB)) \}
\end{verbatim}

The following formulas

\begin{align*}
F_1 &= \text{SUBJECTS(AI)} \\
F_2 &= \text{RESEARCHER(name:Smith, fields:SUBJECTS(AI, KB))}
\end{align*}

are not included in I under c, and thus they are not satisfied by I under the conventional first-order LP notion. However, the resulting sets of ungrouping them are included in I under c, as

\begin{align*}
\sigma F_1 \subseteq \sigma I \quad \text{and} \\
\sigma F_2 \subseteq \sigma I.
\end{align*}

Under the HILOG-R semantics they are satisfied by I. As another example, consider two sets of atoms M_1 and M_2,

\begin{verbatim}
    M_1 = \{ \text{SUBJECTS(AI)} \} \\
    M_2 = \{ \text{RESEARCHER(name:Smith, fields:SUBJECTS(AI, KB))} \}
\end{verbatim}

Intuitively M_2 seems "bigger" than M_1, but they may not be comparable under either c or \(\subseteq\) directly. However, regardless of their original containment relationship, their ungrouped form

\begin{align*}
\sigma M_1 &= \{ \text{SUBJECTS(AI)} \} \\
\sigma M_2 &= \{ \text{RESEARCHER(name:Smith, fields:SUBJECTS(AI))} \}
\end{align*}

can be compared under the unique partial-containment relationship, as

\[ \sigma M_1 \preceq \sigma M_2. \]

With these notions the HILOG-R semantic framework has the following extenstions to the conventional first-order LP model theory:

- The satisfaction of a formula F by an interpretation I is extended from \(F \in I\) to \(\sigma F \subseteq \sigma I\).
- The comparison of two models \(M_1\) and \(M_2\) is extended from \(M_1 \subseteq M_2\) to \(\sigma M_1 \subseteq \sigma M_2\).
- The model intersection expression is extended from \(\cap [M_i]\) to \(\cap : \{ e \cdot \sigma M_i \}\).
- The function for fixpoint computation is extended from \(T_F(I) = I \cup \{ H, H \leftarrow B_1 \ldots B_n \} \) and \(B_i \in I\) to \(T_F(I) = e \cdot \sigma I \cup e \cdot \sigma \{ H, H \leftarrow B_1 \ldots B_n \} \) and \(\sigma B_i \subseteq \sigma I\).
- Based on the above model comparison notion, for a program P there exist a set of minimal models with the same ungrouped form which is referred to as the least model equivalent class \(E_p\). The common ungrouped form of all the models in \(E_p\) is called the immediate least model and is unique. Formally speaking, MM is the immediate least model iff for any model N, \(\sigma N \subseteq MM\). Further, \(\forall M \in E_p\) \(\sigma M = MM\). For a first-order program without set-terms, the least model equivalent class is reduced to a unique element MM and \(\sigma MM = MM\).

HILOG-R is semantically analogous to the canonical LP theory. The above notions are just the generalization of the corresponding notions in first-order LP, and the latter are special cases of the former. Such an analogy can be kept due to that the extension is made at the fundamental set theory level.

3.3 An Example

An example HILOG-R program \(P_1\) is given below showing the reasoning of nested relations. This program concerns the 1-N join where one join attribute is of atomic type and the other is of set-type, which essentially consists in the reconstruction of the nested relation PROJECTS from existing relations PROFS and STUDENTS whose schemes were given before. Figure 3 ill-
lustrates this example.

<table>
<thead>
<tr>
<th>PROJECTS</th>
<th>implementation group</th>
</tr>
</thead>
<tbody>
<tr>
<td>project</td>
<td>name</td>
</tr>
<tr>
<td>NEC90</td>
<td>Smith</td>
</tr>
<tr>
<td>IBM89</td>
<td>Parker</td>
</tr>
<tr>
<td>NASA90</td>
<td>Parker</td>
</tr>
</tbody>
</table>

Figure 3 An HILOG-R Program

\[
\text{PROJECTS} \leftarrow \begin{array}{lll}
\text{design group: RESEARCHERS} & \text{implementation group: PROGRAMMERS} \\
\text{researcher(name: Smith, fields: SUBJECTS(DB, KB))} & \text{programmer(name: John, major: KB)} \\
\text{researcher(name: Parker, fields: SUBJECTS(DB, KB))} & \text{programmer(name: Lee, major: DB)}
\end{array}
\]

\[
\text{STUDENTS} \leftarrow \begin{array}{lll}
\text{student(s#:s101, name: John, major: KB, project: NEC90)} \\
\text{student(s#:s102, name: Won, major: AI, project: NEC90)} \\
\text{student(s#:s203, name: Linda, major: DB, project: IBM89)} \\
\text{student(s#:s207, name: Joe, major: KB, project: IBM89)} \\
\text{student(s#:s305, name: Lee, major: DB, project: NASA90)}
\end{array}
\]

\[
\text{PROFS} \leftarrow \begin{array}{lll}
\text{prof(name: Parker, fields: SUBJECTS(DB, KB), projects: CONTRACTS(IBM89, NEC90))} \\
\text{prof(name: Smith, fields: SUBJECTS(AI, KB), projects: CONTRACTS(NASA90, IBM89))}
\end{array}
\]

\[
\text{4. Conclusions}
\]

The key to database and knowledge base integration consists in reasonable grouping and intimate coupling related data and knowledge. In this paper we have presented an approach for linking subject-related data and knowledge through derived nested relations, and developed a formal framework of nested relation based knowledge representation to support this approach. Since our framework is characterized by extending the underlying notions at set theory level, compared with the previous LP systems for reasoning with complex objects it has richer expressive power, is more canonical and analogical to first-order LP framework.

\[
\text{References}
\]


