Managing Grid Schemas Globally

Kimio Kuramitsu[1]

CPDC, Kogakuin University
kuramitsu@cpd.kogakuin.ac.jp

Abstract. Sharing schemas is a shortcut to data interoperability, while in grid environments there are many difficulties such as schema disagreements and schema evolutions. We propose a new "mappings first, schemas later" schema model, named Grid Schema. The Grid Schema uses the idea of context-free mapping to modularize schemas and its translation rules. This is incorporated into its schema validation mechanism, which enables us to check the compatibility of different formed data. We show the flexibility of the Grid Schema in maintaining global schemas in a distributed, evolving, and multi-cultural environment.

1 Introduction

Data Grid has recently attracted an emerging attention to both scientific and business community. In their fields the amounts of data are increasing up to more than hundreds of terabytes [7]. It is too costly to maintain all the data in a single storage site. A sharing and delivery of the data in a Grid way could be a promising approach to the data explosion.

We consider *RFID Database* as a motivating example of Data Grid. RFID tag is an electronic tag, attached to the real-world object. Using RFIDs as an alternative to the barcodes, products can be uniquely identified. However, the length of the RFID identifier is at most 128 bits, so that it doesn't carry much information. Accordingly, RFID systems must work together with RDBMSs, or other database systems.

Manufacturers, intermediaries (brokers), resellers, or even customers would want to record several information associated with RFIDs. For simplicity, we assume these records are stored in relations, such as $R(\mathsf{rfid}$, attr1, attr2, attr3, ..). A problem is that we are not going to agree on the schemas of these records (attributes). The schema varies from companies to companies or from products to products. It is impossible to assume a unified culture (or a language) in the schemas, because not a few products move across the national borders. Figure 1 shows an example of the movement of books that are stored in Data Grid. We need a transparent means of work together such distributed database in Data Grid, while Data Grid has traditionally focused on the effectiveness in data replica and data delivery.

We are developing the Grid Schema repository. The Grid Schema uses the idea of context-free mapping to modularize schemas and its translation rules. This is incorporated into its schema validation mechanism, which enables us

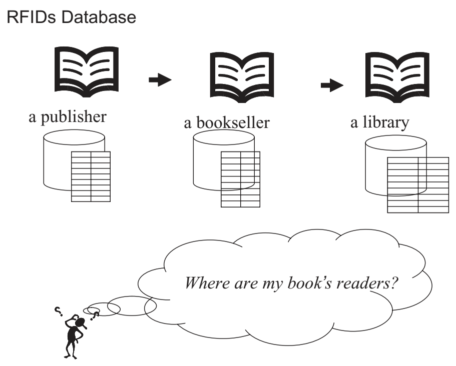

Fig. 1. Motivating Example of RFID Databases

to check the compatibility of different formed data. The management of different forms and representations help us evolve the schema repository as the grid environment changes.

The database community has long discussed schema mediation techniques [1, 4–6, 8, 11, 13, 15]. That is, the schemas are allowed freely, and then maps are created to mediate between conflicted schemas. A rich history of this area has showed that creating maps is not easy, especially when the conflicts are *semantically* complicated. We consider that the mediation approach is unreasonably *inefficient* since data integration and translation are everywhere in Data Grid. That is why we have chosen a reverse approach, say, we first give vocabularies and their mappings to the users, and then the users create schemas over the vocabularies.

This paper models the Grid Schema and discusses the sharing of schemas in Data Grid. The rest of the paper proceeds as follows. In Section 2, we describe our basic ideas underlying Grid Schema. In Section 3, we formalize Grid Schema Model with a type-checking algorithm for schema resolution. In Section 4, we sketch the repository of Grid Schema, which is being implemented on top of RDBMS. In Section 5, we discuss the flexibility of Grid Schema in terms of the management of global schemas. In Section 6, we conclude the paper.

2 Context-Free Mapping

The idea underlying Grid Schema is "mappings first, schema later" – we share mappings before we create schemas. That is, the schemas are defined over *schema parts*, which are given with mappings to interoperate each other. A key is how to modularize the schema parts with keeping the reusability of its mappings. In this section, we informally introduce the idea of *context-free* mappings, which make the core of the Grid Schema model.

2.1 Modularization

The same information can be formed or represented in different ways – which causes the *schema conflict problem* [9, 12, 16–18]. We start by focusing on some conflicts between two simple data elements, say, labeled values. Here are two data elements, excerpted from different sources.

[TEMPERATURE, 40]   [温度, 4]

Here we assume that these elements represent the same information, although they have different labels (we mean that the label "温度" represents temperature in Japanese) and different value representations (40F = 4C).

Schematic (or semantic) conflicts like the above very often occur between different databases. To reconcile the conflicts, we usually use the mapping to translate into a unified data representation. For example, we can specify a mapping rule between elements: the value X of TEMPERATURE $\longmapsto$ [温度, Y], where $Y = (X - 32) \times 5/9$.

The mapping rules are specified after the conflicts are identified. A problem with such rules is less reusability. Let suppose other similar elements, labeled as temperatur and temperatura instead of temperature. In each of different sources, we need to specify new rules, in spite of the fact that the rules might be the almost same. Note that the measurement of units would differ in each labeled element.

To improve the reusability, we focus first on the conversion between Fahrenheit and Celsius. This conversion is universal, or *context-free*, regardless of how the values are labeled. This means that the conversion of values *can be* independent of labels. Thus, we can compose the mapping rule between the elements by two discretely defined mappings:

- *Label mapping*: TEMPERATURE $\longmapsto$ 温度
- *Domain mapping*: 40 $\longmapsto$ 4 (by $Y = (X - 32) \times 5/9$)

We call the value conversion *domain mapping*, while we call the rest *label mapping*, known as the inter-schema correspondence. We assume that all parts in the Gird Schema are modularized in a way that they are mappable only by two types of context-free mappings.

It is important to note that labels are statically associated with *domains* in traditional data models; for example, in the relational model the domain of an attribute TEMPERATURE is determined by *dom*(TEMPERATURE). However, this is very problematic when we want to separate the specification of the label and domain mappings. Not all ones in the Grid use the same domain to represent values in TEMPERATURE. That is, we need a similar separation of domain semantics from the label on the data representation.

In the Grid Schema, we assume that the labels do not *a priori* involve any types of domains. Instead, domains should be given explicitly in an independent form, like [TEMPERATURE, $Fahrenheit$] or [TEMPERATURE, $Kelvin$].

Definition 1 (labels and domains). *We assume (in Grid Schema, defined later) that the structure of the data element is represented by two schema parts:*

- label *to identify the context of the element (what meaning the element carries), and*
- domain *to identify the domain of the element values (how to represent values).*

2.2 Limitations

The separation of the label and domain mappings is trivial when we look at the correspondence between only two data elements. But, when we have the broader scope for element sets, there are two types of schematic conflicts,[1] whose reconciles are not so simple. To keep the context-free modularity, we impose two limitations on the expressiveness of the Grid Schema.

Limitation 1. All labels have the first-class semantics.

Label mappings are not always context-free. This occurs when the meaning of a label depends on other elements. Let consider the following example.

phone-type: office	home-phone:
phone: 03-3812-2111	office-phone: 03-3812-2111

The meaning of phone depends on the value of phone-type, and then we cannot statically determine whether the phone should be mapped into office-phone/home-phone. Otherwise, we need a complex mapping rule, like phone $\longmapsto$ office-phone iff phone-type = office. Fortunately, we can carefully avoid such conflicts in modeling the data. To keep all label mappings context-free, we assume that all labels have the first-class semantics.

Limitation 2. Functional dependency has to be given within a source. Some label mappings need additional value (not domain) mappings. This mainly occurs when two or more elements are *isomorphic*. For example, consider the mapping from the departure element to nights. The scheme of domain mappings does not support the value mapping 2004-12-25 $\longmapsto$ 7.

arrival: 2004-12-19	check-in: 2004-12-19
departure: 2004-12-25	nights: 7

In the isomorphic conflict, it is not easy to avoid one of the occurrences because neither is definitive. We believe that label and domain mapping should be disparate in between different sources. To achieve our belief, let us suppose that the label check-out is defined in the right source. The label mapping departure $\longmapsto$ check-out is context-free, and then we can derive the value of nights from the functional dependency if it is formulated by nights = check-out − check-in. In this paper, we assume that we well formulate functional dependencies among labels within a source.

3 Grid Schema Model

In representing data on the Data Gird, there are many formats available, ranging from relational data to XML. For simplicity, we assume that the all the data can be model by a flat relational structure including RFID, which would like $R(\mathsf{RFID}, attr1, attr2, ..)$. Here we highlight how to model a set of *attr*s (without RFID) in each of the data structure.

[1] We consider that one-to-many correspondence like name and (firstname, lastname) can be dealt with by extending domains to model multi values or a complex object.

3.1 Vocabulary

We start by defining the most fundamental schema part, a *vocabulary.* The vocabulary is the common structure to share the definition of both labels and data values (domains).

Definition 2. *A vocabulary is a finite set of terms, denoted* $V = \{v_1, v_2, .., v_i, v_j, ..\}$, *where there exists no two terms* v_i, v_j *that satisfy the mappings:* $v_i \longmapsto v_j$ *and* $v_i \longmapsto v_j$. *(That means the vocabulary has no inner-mappings.)*

Each vocabulary has a unique name, which we identify by the set name V. To distinguish the type of vocabularies, we also use S to denote a label vocabulary, and D for a domain vocabulary. In only domains, intensional set definitions and numerical terms are both allowed, like D_{name} or D_{USD}.

Example 1. The following D_1 is a vocabulary, while D_2 is not a vocabulary because we say fall $\longmapsto$ autumn (and autumn $\longmapsto$ fall).
$D_1 = \{\text{spring}, \text{summer}, \text{fall}, \text{winter}\}$
$D_2 = \{\text{spring}, \text{summer}, \text{fall}, \text{autumn}, \text{winter}\}$

The terms are *not simulatable.* That is, we are not certain which vocabulary a given term should belong to. For example, a term spring may belong to D_1, or other domains (as a mechanical part or a kind of water pool). We denote $D.v$ (or $S.l$) to explicitly represent the value v of D (or the label l of S).

A mapping is defined between two terms in different vocabularies. We write $D.v \longmapsto D'.v'$ for a rule that we can map $D.v$ into $D'.v'$. Similarly, we can map like $S.l \longmapsto S'.l'$. (We may map between labels and domain values, although it makes no sense.) If the mapping from D to D' is total, we simply write $D \longmapsto D'$.

All mappings are assumed to be *context-free*, as we described in Section 2. This is formalized below.

Definition 3 (context-free). *Suppose [S.l, D.v].*

- *If* $S.l \longmapsto S'.l'$ *is given, then we say [*$S'.l'$*, D.v] for arbitrary D.v.*
- *If* $D.v \longmapsto D'.v'$ *is given, then we say [S.l,* $D'.v'$*] for arbitrary S.l.*

Note that we view the semantics of these mappings as a connection to *relative information capacity* [14]. Intutively, the mapping $D.v \longmapsto D'.v'$ means $D.v$ is more informative than $D'.v'$. Also, we can rewrite $D'.v' \sqsubseteq D.v$ in terms of the capacity. We say the equivalence $D'.v' \equiv D.v$ if and only if $D'.v' \sqsubseteq D.v$ and $D.v \sqsubseteq D'.v'$. The equivalence is used to normalize mappings in a storage, which will be described in Section 4.

3.2 Data Representation

In the Grid Schema system, all data are represented over terms defined in vocabularies. We define the data object as follows.

Definition 4 (data). *The data (object) O is a finite set of data elements. Each data element is denoted as* $[S.l, D.v]$, *where the first term is a label in* S *and the last is a domain value in* D. *We presume that the order of the elements is meaningless.*

Example 2. The data can be represented over different label vocabularies or part of each vocabulary. Suppose two label vocabularies S_1 and S_2 are given:

$$
\begin{aligned}
S_1 &= \{\mathsf{title}, \mathsf{author}, \mathsf{isbn}, \mathsf{publisher}, \mathsf{year}\} \\
S_2 &= \{\mathsf{seller}, \mathsf{listprice}, \mathsf{salesprice}, \mathsf{salesdate}\} \\
O_1 &= \{\ [S_1.\mathsf{title}, D_{EN}.\text{"Harry Potter and the Order of the Phoenix"}], \\
&\quad\ [S_1.\mathsf{author}, D_{name}.\text{"J. K. Rowling"}], \\
&\quad\ [S_1.\mathsf{isbn}, D_{isbn}.\mathtt{043935806X}], \\
&\quad\ [S_2.\mathsf{listprice}, D_{USD}.\mathtt{30.00}], \\
&\quad\ [S_2.\mathsf{salesprice}, D_{USD}.\mathtt{12.00}]\ \}
\end{aligned}
$$

We introduce the class C as a schema component. We use the class not only as a metadata to the underlying relational schema, but also as the view specification of a database application.

Definition 5 (class). *The class C is a finite set of attributes, each of which is a pair of label and domain name, denoted $(S.l, D)$.*

Example 3. The following C_1 is a class definition for the object O_1 above.
$$
\begin{aligned}
C_1 = \{\ &(S_1.\mathsf{title}, D_{EN}), (S_1.\mathsf{author}, D_{name}), (S_1.\mathsf{isbn}, D_{isbn}), \\
&(S_2.\mathsf{listprice}, D_{USD}), (S_2.\mathsf{salesprice}, D_{USD})\}
\end{aligned}
$$

We write $C.O$ if the data O is an instance of C, (that is, for each attribute $[S.l, D]$ in C the object O has the corresponding data element $[S.l, D.v]$.)

3.3 Type-Checking

Type-checking is of increasingly importance to data processing over the Data Grid. Specifically, a grid application needs to validate the structure of remote data (in distributed databases) before we process them. Here we should note that the remote data would differ from a schematic view that the application expects. The structural type-checking might reject most of the remote data.

The Grid Schema provides an advanced type-checking mechanism that is incorporated with mappings into the validation scheme. The underlying idea is the *compatibility*. Intuitively, we say that D_{USD} and D_{yen} are compatible because their values are mappable with each other. We apply this idea to check whether a given object is compatible to the Grid view C'.

Definition 6 (mapping-enhanced type-checking). *An data object $C.O$ conforms to C' if and only if:*

- *$C.O$ is an instance of C' ($C = C'$), or*
- *there exists a data instance $C'.O'$ that can be derived from $C.O \longmapsto C'.O'$.*

It is unrealistic of course that we specify mappings between all possible classes. The strength of Grid Schema is that we can check $C.O \longmapsto C'.O'$ from label mappings and domain mappings that are already defined. The algorithm is: $C.O) \longmapsto C'.O'$ is said if for each element $[S_i, l_i, D_j.v_j]$ in O there exists $[S'_i.l'_i, D'_j.v'_j]$ in O', such that $S_i.l_i \longmapsto S'_i.l'_i$ and $D_j.v_j \longmapsto D'_j.v'_j$.

Example 4. We suppose the following mappings are given, prior to the schema validation.

S_3. 書名 $\longmapsto S_1$.title
S_3. 著者 $\longmapsto S_1$.author
S_3.ISBN $\longmapsto S_1$.isbn
S_3. 定価 $\longmapsto S_2$.listprice
S_3. 定価 $\longmapsto S_2$.salesprice
$D_{USD} \longmapsto D_{yen}$

The following object O_2:

$O_2 = \{$ [S_3. 書名 , D_{EN}."Harry Potter and the Order of the Phoenix"],
[S_3. 著者 , D_{name}."J. K. Rowling"],
[S_3.ISBN, D_{isbn}.`043935806X`],
[S_3. 定価 , D_{yen}.2800],
[S_4.TYPE , $D_{JISXB01}$.`EnglishBook`] $\}$

is compatible to C_1, because we can translate it to $C_1.O'_2$:

$C_1.O'_2 = \{$ [S_1.title, D_{EN}."Harry Potter and the Order of the Phoenix"],
[S_1.author, D_{name}."J. K. Rowling"],
[S_1.isbn, D_{isbn}.`043935806X`],
[S_2.listprice, D_{USD}.25.00],
[S_2.salesprice, D_{USD}.25.00] $\}$

4 Grid Schema Repository

This section reviews implementation issues on the Grid Schema repository, and its inference engine.

4.1 Overview

The Grid Schema repository delivers vocabularies and their mappings across the Grid. Figure 2 shows an architectural overview of the Grid Schema repository. For simplicity, we assume that the repository is centralized. (We do not consider the scalability and the locality issues in this paper.) The Grid Schema repository works together the OGSI-compliant services, such as GSI and MDS.

In designing grid databases, grid organizations first create the Grid Schema classes by choosing a collection of labels and domains from the repository. In cases of the lack of necessary vocabularies, they are allowed to add new vocabularies into the repository. In the end, all the classes are published as part of the Grid metadata in OGSI Monitoring and Discovery Services.

In accessing grid database, grid applications look for remote databases by using MDS. They check their class, compared to these database classes. If matched, remote data are moved and converted to the local database, or queries are translated into remote databases. Since the Grid Schema repository provides only mappings for the interoperability of grid databases, the efficiency of them in querying is beyond the scope of this paper.

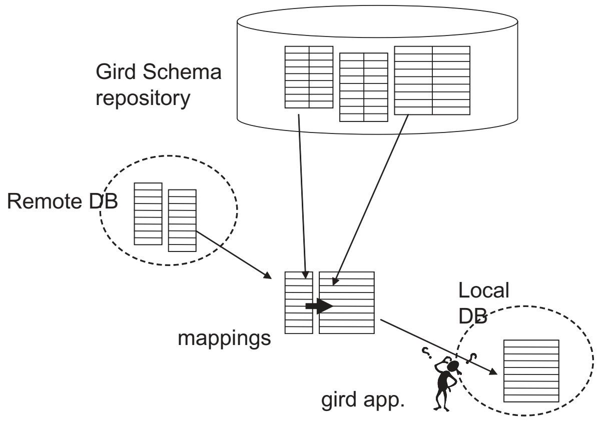

Fig. 2. An Architectural Overview of the Grid Schema repository

4.2 Storing and Retrieval of Mappings

The Grid Schema repository must store huge volumes of terms and mappings. The advantage of the Grid Schema is that the model is so simple that we can easily maintain vocabularies on RDBMS. Intuitively, we can store all mappings in a relation; the mapping $D.v \longmapsto D'.v'$ can be rewritten by a relation $\mathsf{T}(D, v, D', v')$.

More importantly, we can derive new mappings over existing T relations. The inference rules, supported in the Grid Schema, reflect the set and mapping theory (reflexivity and transitivity).

Definition 7 (inference rules). *We use T to denote the derived relations.*
I1. *$T(D, v, D, v)$ for all terms in the repository, and*
I2. *$T(D_1, v_1, D_3, v_3)$ if $\mathsf{T}(D_1, v_1, D_2, v_2)$ and $\mathsf{T}(D_2, v_2, D_3, v_3)$*

Although we represent mappings by T relations on RDBMSs, there would be many tips necessary to implement effective storage and mapping retrieval. Due to space constraint, we only describe the outline of our Grid Schema repository.

– *Normalization.* We classify all terms in the Grid Schema into normal vocabularies by equivalence class over $\equiv$. Normal domains are partially ordered over $\sqsubseteq$ in the whole Grid Schema (by Definition 2).
– *Partitioning.* We partition T relations by each of domain/range in mappings. The partitioned tables are called domain tables. (Partitioning table is a well-known technique to improve the selection operation.)
– *View Maintenance.* The inference rule **I2** is implemented with views that are equijoined over domain tables.

5 Discussion

The Grid Schema works as a sort of *global schemas*. Building global schemas is one of the most straightforward means for data interoperability. In practice, however, it is awfully hard to maintain such schemas with keeping good interoperability over a distributing computing environment – where participants desire differently and schemas must update frequently.

Here we discuss the flexibility of the Grid Schema – how the Grid Schema helps us maintain global sharing, compared to existing schema sharing models, such as object-orientated (OO) models. It is important to note that the interoperability in global schemas depends not only on its schema model, but also on how to administrate schemas in a distributed manner. With an attention to the administration, we carefully set the following four situations (A1 $\sim$ A4).

– *A1. (Monolithic).* A central administrator is only allowed to design and modify global schemas. (e.g., EDIs).
– *A2. (Class-Subclass).* The schemas are developed in a hierarchal way. That is, local users are allowed to customize their own subclass schemas from super-class schemas that have been already published. (e.g., some practices under UML)
– *A3. (Object-Subclassing).* A significant variation of A2 is considerable in XML; the schema is virtually made by sub-classing on an instance of data. A typical example can be shown in XML namespaces where we combine an XML document with multiple XML schemas.
– **A4. (Grid Schema)**. All users create their schemas, based on vocabularies in the Grid Schema. In addition, they are free to add new vocabularies to the repository if they specify mappings to existing related vocabularies.

Several difficulties in maintaining global schemas occur in cases of requirement mismatches and schema evolutions. Now we highlight three significant challenges (B1 $\sim$ B3), which very often appear in the administration of global schemas.

– **B1. (Evolving Element)**. Schemas must be evolved, when new requirements arise in the user side. For example, we imagine a brand-new travel frequency program, where some participants want to share the information, while others outside may not like to add modifications in the global schema.
– **B2. (Value Flexibility)**. It is important to allow representational variations on the Web environment, because all values can be not necessary standardized. For example, US people and European would not reach an agreement on how to write temperature in a unified way (i.e., Celsius vs. Fahrenheit). The value flexibility helps a rapid consensus building in the agreement.
– **B3. (renaming/localization/refactoring)**. A long-term use of global schemas requires the management of different naming schemes, ranging from a minor modification of errata naming to an extensive refactoring in the schema revision and merge. In addition, we have to make schemas internationalized, because labels in querying are a significant part of developing applications.

Table 5 summarizes our evaluation on the flexibility. The `UPDATE` column represents whether the schema update is centralized (C) or decartelized (D).

We begin with a general comment. In grid environments, the decentralized update is more desirable than the centralized one, since it doesn't limit chances in exploiting new data applications and services. In this light, the Grid Schema allow users to update their vocabularies, even though it runs on a central site.

Next, we take a look at each challenge. For convenience, we write $C \prec C'$ for the IS-A relationship in the OO model, that means C' *is a (subclass of)* C.

Table 1. Matrix of Agreement/Flexibility

	UPDATE	B1	B2	B3	safe
A1	C	o	x	x	o
A2	D	x	x	x	x
A3	D	o	x	x	x
A4	D	o	o	o	o

B1. (Evolving element). Suppose we want to add an element traveler-info to the existing schema. In the class-subclass style, the schema is created and updated in an incremental way (e.g., adding a new class C'' (that extends the traveler-info) between C and C', such that $C \prec C'' \prec C'$). Note that the modification of the class hierarchy may cause a name conflict between C'' and C'. In a case of A1, such a name conflict can be controlled by a single naming authority. In cases of A3 and A4, we can use namespaces to distinguish existing names from a new name.

B2. (Value flexibility). Suppose we don't want to fix currency in price. In the class-subclass style, we usually define an abstract type $D_{currency}$, and then use its extended types, such as D_{usd} and D_{yen}. The variation is controlled by $D_{currency} \prec D_X$ – whether D_X is a subtype of $D_{currency}$. The problem is that the extended subtypes are not interoperable with each other. To sketch, suppose a grid application that only takes the values of D_{usd}. The application is not able to deal with values of D_{yen} even if values are validated by $D_{currency} \prec D_{yen}$. In contrast, the Grid Schema provides the mapping interoperability between D_{usd} and D_{yen}, and then allows the agent to understand D_{yen} by $D_{yen} \longmapsto D_{usd}$.

We consider that the semantics of mappings in the Grid Schema is more expressive than that of IS-A relation. The comparison is sketched in Figure 3.

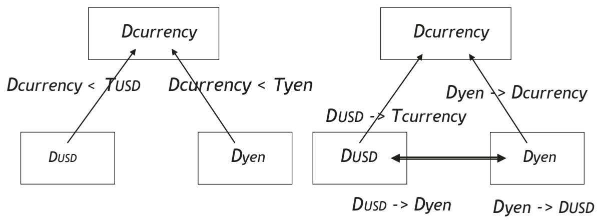

Fig. 3. Comparison of IS-A vs. Mapping

B3. (renaming/localization/refactoring). Suppose some schemas of Japanese version need to run together with English schemas. It is very hard (or has largely been ignored) to deal with different naming schemes in the OO schema designs. The Grid Schema provides the mapping mechanism to the solution. That is, we can create Japanese vocabularies by just translating the English ones. More importantly, different versions are transparent to grid applications if they are compatible by the Grid Schema type-checking mechanism.

Finally, we would like to mention that the Grid Schema is *safe* in terms of the validation. That is, a grid application can safely use the validated data by the application class. On the other hand, the class-subclass extension is not safe in a sense that the extended types, although validated, are not always available in the application. The *polymorphism* may cause heterogeneous data at the representation level.

6 Conclusion

Sharing schemas is a shortcut to data interoperability, while in grid environments there are many difficulties such as schema disagreements and schema evolutions. We propose a new "mappings first, schemas later" schema model, named Grid Schema. The Grid Schema uses the idea of context-free mapping to modularize schemas and its translation rules. This is incorporated into its schema validation mechanism, which enables us to check the compatibility of different formed data. We showed the flexibility of the Grid Schema in maintaining global schemas in a distributed, evolving, and multi-cultural environment.

In this paper, we assume that all grid databases share global object identifiers (such as RFID). In the future, we will extend the sharing schema model to cover more generic data models for scientific databases and XML. In addition, our Grid Schema repository is developing for further experimental study. We would like to discuss the effectiveness, the scalability, and the extensibility of the repository through the development.

Acknowledgement

The author would like to thank all attendants in ICSNW2004 for valuable comments.

References

1. S. Abiteboul, S. Cluet, and T. Milo. Correspondence and translation for heterogeneous data. In *Proceedings of 6th International Conference Database Theory - ICDT '97*, pages 351–363, 1997.
2. S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases.* Addison-Wesley Publishing Company, 1995.
3. D. Carlson. *Modeling XML Applications with UML: Practical e-Business Applications.* Addison-Wesley, 2001.
4. T. Catarci and M. Lenzerini. Interschema knowledge in cooperative information systems. In *Proceedings of Conference on Cooperative Information Systems*, pages 55–62, 1993.
5. S. Cluet, C. Delobel, J. Siméon, and K. Smaga. Your mediators need data conversion! In *Proceedings of ACM SIGMOD International Conference on Management of Data - SIGMOD98*, pages 177–188, 1998.
6. S. B. Davidson and A. S. Kosky. Wol: A language for database transformations and constraints. In *Proceedings of the 13th International Conference of Data Engineering*, pages 55–65, 1997.
7. I. Foster, and C. Kesselman. *The Grid : Blueprint for a New Computing Infrastructure* . Morgan Kaufmann, 1998.

8. C. H. Goh, S. Bressan, S. Madnick, and M. Siegel. Context interchange: New features and formalisms for the intelligent integration of information. *ACM Transactions on Information Systems*, 17(3):270–293, 1999.
9. V. Kashyap and A. P. Sheth. Semantic and schematic similarities between database objects: A context-based approach. *VLDB Journal*, 5(4):276–304, 1996.
10. D. Lee and W. W. Chu. Comparative analysis of six xml schema languages. *Sigmod Record*, 29(3):76–87, 2000.
11. A. Y. Levy, A. Rajaraman, and J. J. Ordille. Querying heterogeneous information sources using source descriptions. In *Proceedings of the 22nd VLDB Conference*, pages 251–262, 1996.
12. W. Litwin, L. Mark, and N. Roussopoulos. Interoperability of multiple autonomous databases. *ACM Computing Surveys*, 22(3):267–293, 1990.
13. J. Madhavan, P. A. Bernstein, and E. Rahm. Generic schema matching with cupid. In *Proceedings of 27th International Conference on Very Large Data Bases – VLDB 2001*, pages 49–58, 2001.
14. Renée J. Miller, Yannis E. Ioannidis, and Raghu Ramakrishnan. The use of information capacity in schema integration and translation. In *Proceedings of 19th International Conference on Very Large Data Bases*, pages 120–133. Morgan Kaufmann, 1993.
15. T. Milo and Z. Zohar. Using schema matching to simplify heterogeneous data translation. In *Proceedings of 24th International Conference on Very Large Data Bases – VLDB 1998*, pages 122–133, 1998.
16. A. M. Ouksel and A. P. Sheth. Semantic interoperability in global information systems: A brief introduction to the research area and the special section. *SIGMOD Record*, 28(1):5–12, 1999.
17. E. Pitoura, O. Bukhres, and A. Elmagarmid. Object orientation in multidatabase systems. *ACM Computing Surveys*, 27(2):141–195, 1995.
18. A. P. Sheth and J. A. Larson. Federated database systems for managing distributed, heterogeneous, and autonomous databases. *ACM Computing Surveys*, 22(3):183–236, 1990.
19. V. Vianu. A web odyssey: from codd to xml. In *Proceedings of Symposium on Principles of Database Systems*, pages 1–15, 2001.