Joint Conference of 1996 International Computer Symposium
December 19~21, Kaohsiung, Taiwan, R.O.C.

SLOODS: A Self-Learning/Organizing Object-Oriented Database System Based on ObjectStore

Shian-Hua Lin and Yueh-Min Huang
Department of Engineering Science, National Cheng Kung University, Tainan, Taiwan, R.O.C.
E-Mail: tom@system.es.ncku.edu.tw

Abstract

*In this paper, we propose a system, Self-Learning/ Organizing Object-oriented Database System (**SLOODS**), to embed object-oriented design in an integrated intelligent database system. To discover knowledge from the database, an automatic inductive self-learning mechanism for OODB (Object-Oriented Database) is designed to support the function. Then the knowledge, stored in the persistent knowledge object network (KON), is further used to self-organize the database schema to adapt the versatile information. In this way, the induced knowledge in KON can be reused to create KBS or semantically optimize the database query. Thus, the system can be applied to develop applications of DBS and KBS simultaneously.*

1. Introduction

Database systems (DBS) and Knowledge-based systems (KBS or Expert systems) have developed for a long time. From the functions of DBS and KBS, the former is used to maintain and access a large amount of data efficiently, and the latter is used to infer new knowledge from existing knowledge and data. The knowledge (schema or data model) supports DBS to maintain and access data, while the data (facts in working memory) support the inference of KBS. Thus, the concepts for data and knowledge are covered for each other so as the DBS and KBS.

The weakness of DBS is the maintenance of data consistency and the search of desired information in a very large database. For the consistency problem, some systems [8] provided users to specify constraints or rules in the schema for improving the semantics of data model. Also, the problem of query optimization can be processed by such knowledge. However, the specified rules can not be modified for adapting the dynamic or temporal change of the database. For example, the new and correct data may violate the original schema so that it can not be inserted into the database unless we reconstruct the database again. Recently, many OODBs provided the function of schema evolution [21,23] to dynamically change the database schema. However, DBA/DBI (database administrators/ implementers) still needs to spend efforts to write the evolution code. It means that such DBS can not reorganize the schema automatically to accommodate the evolution of the database. As for the search problem, conventional DBS use indices to indicate the search from large data files. Unfortunately, indexed attributes are not always involved in queries and DBS cannot make indices for each attribute due to the complexity for maintaining the consistency. Thus, the conventional DBS techniques revealed the deficiency in the future information system.

On the other hand, KBS also suffers from the overhead of communication between knowledge engineers and experts during the phase of knowledge acquisition. Hence, many learning systems based on DBS have been developed, such as DBLearn [22], EXPLORA, Knowledge Discovery Workbench [14], etc. On the operational phase, the maintenance and the inference for a very large knowledge base are problems as well. To overcome the difficulty, some systems have applied the DBMS techniques in the maintenance of the large KB [16]. As for the efficient inference, Rete [3] was a typical well-known approach.

Based on those concepts, we develop an intelligent database system on ObjectStore [6,23]. In the rest of the paper, we describe the relative research in section 2. The overview and architecture of the system are specified in section 3. Section 4 contains the implementation of each component of the system, and the design flow of the system's application is introduced in section 5. Finally, the conclusion and future work are stated in section 6.

2. Related works

Recently, many investigations have focused on coupling KBS and DBS [20]. However, most of those investigations do not take the advantages from each other to a certain extent. Basically, those studies can be classified into three types:

1. *Extending Database Management System* [8, 11]: One approach is to embed the logic programming concept into the DB programming language to form the deductive DB. The other is to apply the inference concept of trigger-action in DBS and to construct an active DB such as POSTGRES [12]. Consequently,

those systems are merely enhanced rather than integrated.
2. *Extending Knowledge-based System* [16]: Some systems employ the techniques of DBS in KBS. Also, those systems are not tightly integrating systems with KBS and DBS.
3. *Integration of KBS and DBS* [13]: The system usually integrates KBS and DBS by the object-oriented approach to accomplish a VLKBS (Very Large KBS). However, the system never investigates the studies, such as mining knowledge from database for the semantic query optimization. Thus, although the system combines the design of DBS and KBS, it does not integrate both systems in a good manner.

On the aspect of object-oriented database systems, logic predicates or rules have been integrated into databases to form deductive/active object-oriented databases [8, 11]. In such design manners, they lack of the flexibility of dynamic insert/update knowledge. To rectify these drawbacks, SLOODS is designed to take assets from both KBS and DBS to achieve an intelligent and flexible information system.

In our studies, knowledge discovery in database (KDD) acts as a bridge to connect both systems. In KDD, the learning engine regards the database as the training set so that objects in OODB are training instances. In Han's *Attribute-Oriented Induction (DBLEARN)* system [9, 22], the original data (accessed from SyBase) is inductively generalized by using the concept hierarchy defined by the user for each attribute. This generalization and induction process are continued until the number of remaining tuples is less than the defined *table-threshold*.

As for the query optimization, conventional techniques use syntactic knowledge of operations and storage details of relations to optimize queries [18]. Since the syntactic optimization lacks of the entire semantic knowledge of the state of a particular database, in many cases it produces suboptimized queries. For instance, the syntactic optimizer can not detect and eliminate semantically redundant restrictions or joins in a query. Of course, it also fails to introduce semantically redundant restrictions or joins to reduce the overall cost of a query. On the other hand, semantic query processing [10,17] introduces a new perspective to query optimization. Instead of re-sequencing the operators or indexing data, it tries to exploit possible semantically equivalent transformations based on Query Transformation Rules (QTR) of the particular database. QTR is different from (static) semantic constraints in relational models. It is a dynamic state of integrity constraints for a particular database, which are useful for semantic query optimization for universally quantified queries [7]. By applying KDD in SLOODS, induced knowledge will completely represent the current state of a particular database; thus it is suitable to be applied in semantic query transformation.

3. Overview of SLOODS

Based on the persistent programming environment of ObjectStore, SLOODS has been developed, by taking advantages of both KBS and DBS, to achieve an intelligent and flexible information system. First, based on the template, Integrated Knowledge/Data Model (**IKDM**), users can develop the schema of an application. Basically, IKDM is the extension of a generic object-oriented data model to accommodate both knowledge representations (in the format of rules) and DB schema for coupling both DBS and KBS. The kernel component of SLOODS, Adaptive Knowledge/Data Manager (**AKDM**), is developed to organize and manage the schema based on IKDM.

Graphic User Interface (**GUI**) is used to direct the user to construct the schema of database. Between GUI and AKDM, Schema Learning Engine (**SLE**) is used to help the user construct a correct IKDM schema. We employ inductive learning in Inductive Learning Engine (**ILE**) as a bridge to seal the gap between DBS (data) and KBS (knowledge). Applying inductive learning in DBS, implicit knowledge can be mined from the database. After learning knowledge from the database, knowledge is represented as a Knowledge Object Network (**KON**) in AKDM for each application database. The inference mechanism is also embedded in KON. Thus, the user can also develop his expert system based on the learned knowledge, or by constructing/modifying extra/existing knowledge through GUI. Also, knowledge in KON can be applied to optimize database queries by Intelligent Query Engine (**IQE**).

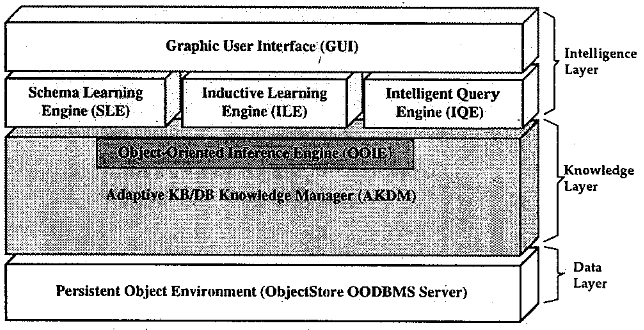
Graphic User Interface (GUI)
Schema Learning Engine (SLE)
Inductive Learning Engine (ILE)
Intelligent Query Engine (IQE)
Object-Oriented Inference Engine (OOIE)
Adaptive KB/DB Knowledge Manager (AKDM)
Persistent Object Environment (ObjectStore OODBMS Server)
Intelligence Layer
Knowledge Layer
Data Layer

Fig. 1. The architecture of SLOODS.

The architecture of SLOODS hierarchically consists of three layers shown in Fig. 1. The top layer is Intelligent Layer that provides users with a friendly interface and replies to users' requests intelligently by those components under GUI. Knowledge Layer, in the middle, is the kernel of the system. AKDM is the key component in the kernel. Each component of SLOODS is executed on ObjectStore's persistent environment, Data Layer at the bottom.

3.1 Intelligence Layer (Self-learning)

On the upper level of Intelligent Layer, GUI is designed to provide an intelligent interface which guides the novel user to construct his object-oriented database schema, database query, and expert system. First, during the construction of schema, the user (DBI) can construct an object-oriented schema for application database according to the message suggested by SLE. The design of SLE is based on the knowledge extracted from analyzing the object-oriented data model [1,2,15] to classify the ill-construction of OO schema. In the study, we observe that properties (attributes) and behavior (methods) in a class can be used to detect such ill-construction. Then, SLE suggest that users adjust the relationship such as generalization/specialization in the class hierarchy or lattice based on the knowledge.

After modeling the schema correctly, users can build the database. ILE is started by DBA or by the system automatically during the system idle time. It extracts implicit knowledge from the database. We have proposed an automatic and efficient learning method applied in ILE to learn knowledge from OODB [4,5]. Through employing the same concept associated with Han's DBLEARN, ILE extends the concept of RDB to OODB as follows:

- Objects in OODDB are regarded as examples.
- Attributes in DB are the features of training instances.
- The generalization/specialization relation in OODB can be treated as the G/S concepts of *version-space* [24].
- Complex objects are partitioned into relevant training sets according to aggregation/association relationships.

After SLE and ILE have supplied schema and inductive rules related to the database, the user can construct the expert system associated with the problem domain of the application database. IQE uses the knowledge to provide a friendly query interface and to perform the semantic (intelligent) query optimization.

3.2 Knowledge Layer (Self-organizing)

AKDM provides the representation/storage/management of knowledge (schema) in this layer. In addition to providing traditional features of OODM, AKDM regards knowledge as objects in the database. Thus, the system can dynamically change knowledge of a class to accommodate the database updating due to inductive learning or user's edition. The data model (IKDM), knowledge representation, is defined as follows:

```
// Definition of Class:
    class <class_name> [:: <class_name>+] {
        [key (<attribute_name> : <primitive_type>
        <Constraint>*)]// Key must be primitive types.
        [attribute (<attribute_name> : <type>    <Constraint>*)]//
     Restricts the attribute domain.
        [aggregation (<attribute_name> : <type> <Constraint>*)+]//
     Aggregation Attribute.
        [method (<type> <method_name> (<argument>*))]
        }
  // Definition of Constraint:
        Constraint ::= [if LHS then RHS]*// Active rules
                     | Range // Restrict the attribute range
  // Definition of Rule:
        [<event>] rule <rule_name> :: <class_name> {
            [RHS] | [ if LHS then RHS ]
            }
// Definition of Method: The same with C++.
// Note: * means zero or more; + means one or more; [ ... ] is optional; <
... > is the non-terminal symbol; ( .. ) means a definition clause.
```

Knowledge base is stored as a complex object net in ObjectStore called *knowledge object network (KON)*, and knowledge inference is transformed into navigation among the network by taking the advantages of ObjectStore's persistent environment. Thus, objects in the network are not only the knowledge repositories but also active agents taking the partial responsibility of inference.

In addition to storing and inferencing knowledge, AKDM also automatically maintains the semantic of database schema, based on knowledge induced by ILE and ObjectStore's *schema evolution* mechanism, to keep the schema up-to-date. We term the schema as *self-organizing schema*. For example in Fig. 2, we can promote the schema up to a richer semantic schema, if the following classification rules are induced.

IF (manufacture = BMW) & (body = streamline) & (Engine = SportsCarEngine) THEN Car is SportsCar.
IF (body = ~streamline) & Engine = SedanEngine THEN Car is Sedan.
IF (body = ~streamline) & Engine = BusEngine THEN Car is Bus.
IF (power = High) & (C.C. = ~Low) THEN Engine is SportsCarEngine.
IF (power = ~High) & (C.C. = ~High) THEN Engine is SedanEngine.
IF (power = High) & (C.C. = High) THEN Engine is BusEngine.

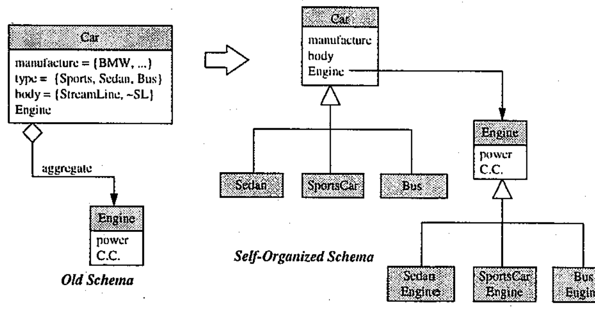

Fig. 2. Self-organizing OODM according to learned rules.

3.3 Data Layer (ObjectStore)

In this layer, currently, we use ObjectStore [6,23] as a testbed for the following reasons:

1. *Persistent programming environment*: ObjectStore provides persistent DB classes, based on its virtual memory mapping architecture, to establish an efficient persistent C++ programming environment. All system components of SLOODS are persistent objects of on it.
2. *Metatype object of schema*: The schema of a database application is also stored as objects in ObjectStore, thus users can get schema information by retrieve these

objects. Naturally, the information is very important to the system. For example, during inductive learning, ILE needs the information of schema such as types of attributes and relationships between classes.

3. *Schema evolution*: ObjectStore's schema is defined by C++ classes. Moreover, It also provides the schema evolution with transformation function specified by users. Thus, AKDM can generate the required functions to achieve the self-organization of schema.
4. *Object version*: The property eliminates the cost of reconstructing knowledge objects if the system and application are required to be updated.

4. The implementation of SLOODS

In this section, we describe the implementation of SLOODS according to components shown in Fig. 1. ObjectStore 3.0.2 is installed in NT Server 3.51 and the system is implemented with Virtual C++ 4.0.

4.1 Schema Learning Engine (SLE)

During the development of a database application, the design of a schema is the critical factor for the quality and the life time of the application. For OODB users, it is not realistic to expect DBI of relational DB to provide an adequate OO schema due to the different concepts in two data models. Thus, we investigate and analyze the properties of OOD [1, 2, 15] and develop rules to detect the ill-construction of OO schema. Those rules are classified into two types shown in Fig. 3.

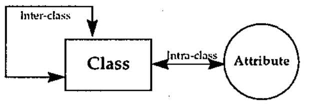

Fig. 3. Relationships among classes and attributes.

Intra-class: The rule revises the ill-relationship between a class and its attribute with M-to-N relationship. If some attributes must be included in the relationship, it should be represented by another class type to keep extra necessary information to reduce the data redundancy and avoid the *update anomaly*. Thus, if an M-to-N attribute is detected in a class, SLE will inquire the user whether extra members are necessary to be included. If they are necessary, the M-to-N attribute is regarded as an attribute, and a new class is generated to link the M-to-N *relating/related* classes. Otherwise, M-to-N relationship remains the same without extra information. The same case is also occurred in ER-schema of relational DB [18]. The rule is applied in the schema construction time.

Inter-class: The inadequate relationships among classes can be detected according to common members:

1. Attributes: name, type, and relation (1-to-1, 1-to-N, and M-to-N).
2. Methods: name, argument type, and return type.
3. Rules: event, LHS, and RHS.

Basically, there are four rules to remedy those inadequate relationships. The first three rules are applied during the construction of the schema. The last one is an example of self-organizing schema and is employed at the run-time of database. Those rules are summarized as follows:

1. Combine two identical classes with the same members.
2. Construct a parent-child link for subsuming classes.
3. Generalize a superclass for classes with common members.
4. If there is an attribute whose values are sparse in a class (i.e., most objects have NULL values for the attribute field), specialize a subclass which contains the attribute for those objects to reduce the data redundancy.

According to the relevance between those heuristic rules, the rule of *intra-class* should be applied in advance since the decision of *inter-class* rules depends on the class's members. While members of intra-class are consistent, similar members of inter-class can be discovered in the next step. Since a schema may be constructed by different DBI, the similar member is adopted in the detection. To unify the similarity, the SLE needs to inquire the user to redefine members which are summarized below according to their priorities. Since the attribute may be referred by methods or rules, the highest priority is assigned to it.

1. Redefine name, types, or relationships of those similar attributes so that they are consistent in relevant classes.
2. Inquire the user if the similar method is the same for relevant classes, then select a method as the unified version.
3. After attributes and methods have been unified, the similarity among rules can be precisely determined.

After similar members have been unified, those identical classes are combined. Then the parent-child link can be constructed between two subsuming classes. Finally, a new superclass is generated. The superclass will be inherited by similar classes which have common members after applying the above processes.

4.2 Inductive Learning Engine (ILE)

ILE is implemented according to the generalization/ specialization concept from *version-space* [24] and the entropy of learning attribute from ID3 [19]. The learning method [4,5] is divided into three stages. First, ARCH is used to generate the concept hierarchy of each learning attribute. Next, OGL is used to search an optimal generation level for each attribute's concept hierarchy. Finally, ASE is applied to guarantee the accuracy of induced rules. Due to the limited paragraph, we briefly describe those three mechanisms and the details can be referred in [4].

ARCH (Automatic geneRation of Concept Hierarchy)

ARCH is a systematic method to construct the concept hierarchy of each attribute automatically. There are three kinds of attributes in OODB: numerical (continuous or discrete), symbolic (unstructured), and objectified attributes. The first two kinds of attributes are defined with *primitive* types whose concept hierarchies can be constructed by using the fuzzy theory. The last attribute refers to *non-primitive* type whose concept hierarchy is constructed according to its relation (association, aggregation, or inheritance).

OGL (Optimal Generation Level)

OGL is proposed to improve the performance of ID3 and Version Space by determining the *optimal generalization level* of each attribute according to the entropy. We applied OGL in ID3 (*modified ID3*) and simulated the method with Version Space according to the complexity and accuracy of induced rules in [4]. The result shown in Fig. 4, 5 proves that *modified ID3* effectively reduces the complexity of induced knowledge with only little loss of the knowledge accuracy.

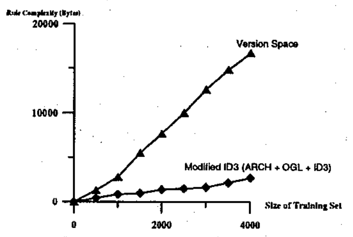

Fig. 4. Simulation result of *modified ID3* (the rule's complexity).

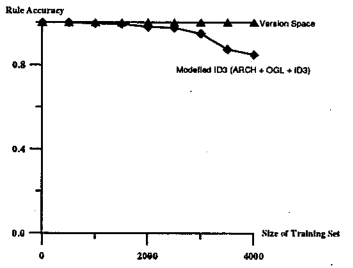

Fig. 5. Simulation result of *modified ID3* (the rules' accuracy).

ASE (Attribute Selection by Entropy)

While learning from a larger training set by *modified ID3*, the accuracy of induced knowledge may become less than 0.8 according to Fig. 5. Thus, another approach, ASE, is proposed to get the trade-off between Version Space (high rule accuracy) and *modified ID3* (low induction cost).
In ASE, the *greedy* and *hill climbing* approaches are used to find optimal classification rules. Entropies calculated in OGL are sorted in an ascending order. Then, the two smallest attributes are selected and combined into a new attribute. If the entropy of new attribute is smaller than the original two attributes, the two attributes are substituted by the new combined one. Otherwise, the third small attribute is selected instead of the second one, and vice versa. Hence, the *greedy* implies that the smallest one is the first choice and *hill climbing* refers to the situation in which the entropy of combined attribute cannot be increased.
The process continues until all attributes have been tried, or the combined entropy is less than the *entropy-threshold*, which is the minimal accuracy (probability) of inductive knowledge input by the user.
Combining with ARCH, OGL, and ASE, our learning process is called *AGE*. Under the same cases, the simulation results shown in Fig. 6. proves that AGE can control the accuracy (probability = 0.9) by the threshold (entropy = 0.469), if the user wants a higher accuracy of induced rules.

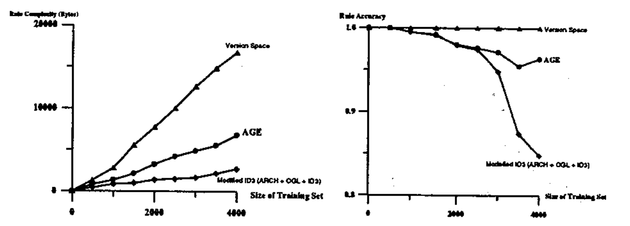

Fig. 6. Comparisons of Version Space, *modified ID3*, *AGE*.

4.3 Intelligent Query Engine (IQE)

In object-oriented database systems, the query acts as the navigation among complex objects rather than the join among tables in RDB. To minimize the navigation cost in the complex tree (or lattice), a semantic query optimizer is required. In SLOODS, IQE supports intelligent query processing based on knowledge in KON, such as constraints of attributes, semantic rules defined by users, and transformation rules induced by ILE.
The query graph in IQE is represented by the tuple of (N, P, E), which corresponds to nodes (classes), predicates in a node, and edges (navigation path) connected to nodes. For Object SQL of ObjectStore, the information corresponds to FROM-clause (classes), WHERE-clause (predicates), and SELECT-clause (destination of the navigation). Predicate is represented by (M1 op M2), where M1 and M2 refer to members (attribute or method) of a class. IQE focuses on predicates and defines its cost model based on the member type and the predicate's selectivity. There are four types of members: SVA/M (Single Value Attribute/Method), MVA/M (Multiple VA/M). Since the method invocation cost is higher than the data retrieval cost, we can define the order of cost: cost(SVA) < cost(MVA) < cost(SVM) < cost(MVM). The selectivity is defined by which operation ($=, >, <, \neq, \geq, \leq$) is applied in the predicate.
Under the assumption of *IO-dominance*, the cost model of a forward navigational query (without any indexes and *path reduction*) shown in Fig. 7 is:

$$\text{Cost}(Q) = \text{Cost}(IO)$$

Assume that objects in database are uniformly distributed, the number of page I/O is linear to the number of objects for the same class. Thus, the I/O cost will be

$|O_i| \times UIO(C_i)$, where $|O_i|$ is the number of the retrieved objects from C_i, and $UIO(C_i)$ is the unit of I/O cost to retrieve an object from C_i. If we scan all objects from C_i, then $|O_i| = |C_i|$. We also assume that the selectivity factor $Sel(R_{C_i})$ of a restriction (predicate) in a class C_i can be estimated from database statistic information. Thus the I/O cost of forward navigation is:

$$Cost(IO) = [|O_1| \times UIO(C_1)] + [|O_2| \times UIO(C_2)] + \ldots + [|O_n| \times UIO(C_n)]$$

$$|O_i| = |C_1| \times \prod_{j=1}^{i-1}\left(Sel\left(R_{C_j}\right) \times rel_{j,j+1}\right),\ i > 1$$

Assume there is a common (virtual) root class C_0 with $Sel(R_{C_0}) = 1$ and with $rel_{0,1} = 1$. EQ (2) becomes:

$$Cost(IO) = \sum_{i=1}^{n}\left\{|C_1| \times \prod_{j=0}^{i-1}(Sel(R_{C_j}) \times rel_{j,j+1}) \times UIO(C_i)\right\}$$

By performing semantic-equivalent query transformation, restrictions in a class R_{C_i} are transformed to R'_{C_i}, and the query cost becomes:

$$Cost(Q') = \sum_{i=1}^{n}\left\{|C_1| \times \prod_{j=0}^{i-1}\left(Sel\left(R'_{C_j}\right) \times rel_{j,j+1}\right) \times UIO(C_i)\right\}$$

The semantically optimized query is the transformed query with minimal $Cost(Q')$, or the original query (if $Cost(Q') > Cost(Q)$, for transformed queries).

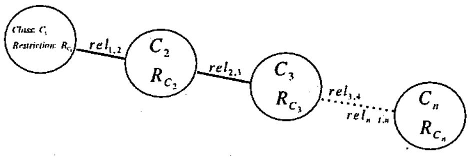

Fig. 7. Chaining navigation path.

After analyzing the cost model of *string* query (forward navigation), the cost of other query types, star and hybrid shown in Fig. 8, can be estimated by being transformed into the summation of costs of several string queries. The remaining problems are to collect semantic rules for transformation and to develop an efficient SQO algorithm. In the beginning, IQE applies knowledge and inference mechanism of KON to obtain all implied predicates based on predicates in FROM/WHERE-clause. IQE uses the greedy approach, by using the forward navigation, to select predicates with minimum cost from root to target class nodes. Once a predicate was selected, IQE checks if the semantic of those selected predicates is the same as the semantically equivalent state is achieved. The process continues subsequently to achieve semantic equivalence.

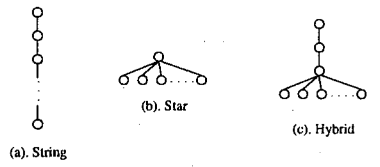

Fig. 8. Types of navigation paths.

4.4 AKDM

Since the quality of schema will affect the life cycle of a database, we propose IKDM as the template to represent the schema and knowledge of the application based on the schema. AKDM manages the database and reorganizes the schema based on the knowledge (KON). Thus, we illustrate KON and self-organization mechanism of AKDM in detail.

KON

Both knowledge and inference mechanism are implemented in the persistent object, KON, in ObjectStore. The object model of KON is shown in Fig. 9. The pattern matching process is partitioned into the recursive sequence: Class → Attribute → Condition Test → Action. Initially, constraints and rules in classes of a schema are inserted into KON. In other words, an application schema of SLOODS corresponds a KON in AKDM. Rules induced by ILE or inserted by users from GUI are also added into the associated application's KON during the run-time.

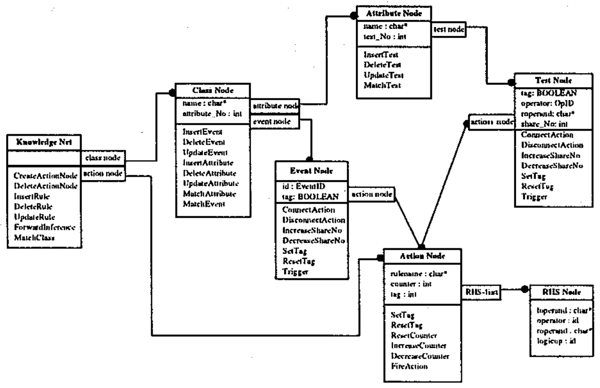

Fig. 9. The schema of KON.

KON takes advantages of the fast matching algorithm, Rete [3], which is efficient for a large KBS due to the same test condition (predicate) is matched only once. Taking the example from Fig. 9, the process of forward Inference behaves is shown in Fig. 10. Hence, the inference process is transformed into the object-navigation in ObjectStore, and both of the knowledge and the inference engine are embedded in KON.

KON includes events in the if-part of the rule, thus the number of predicates (Counter in ActionNode) in a rule equals to the number of events and predicates in LHS. Initially, KnowledgeNet receives facts and then sends them to their relative classes. In ClassNode, it matches those facts (events or predicates) and sends them to corresponding EventNodes or AttributeNodes. The AttributeNode continues the matching process and passes the required information to its related TestNodes if matched. If Event/Test Nodes are matched, they decrease the Counter of their respective ActionNodes. ActionNode checks its Counter until the value is zero (the situation implies that all conditions are satisfied). Again, ActionNode's rule is fired to KnowledgeNet, and the same process is processed until all facts are matched once. SetTag in Event/Test Nodes is used to avoid to decrease ActionNode's Counter again, if Event/Test has been triggered. In this way, the problem of infinite inference due to the cyclic rule chaining is also resolved by SetTag. For instance, if "a → b" and "b → a" form the cyclic chaining. The firing operation of the both rules is guaranteed only once due to SetTag in the TestNode of "a" and "b".

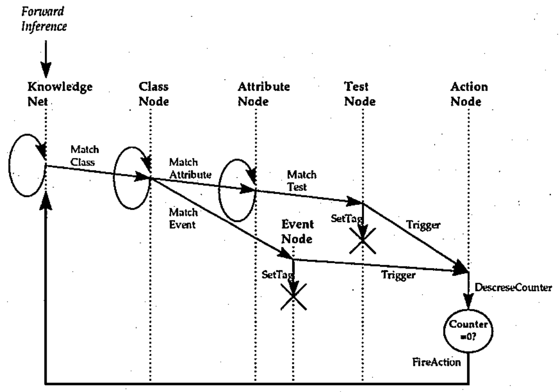

Fig. 10. Forward inference in KON (AKDM).

Thus, by applying KON framework, objects existing in top layer invoke the objects at the lower layer without waiting for the returned results, and the pipeline inference can be achieved by multithread in Windows NT. Furthermore, the inference model is easy to be port in distributed environment for experiments of distributed AI (DAI).

Self-organization

In AKDM, the self-organization of schema is easily achieved by detecting classification rules from KON, and then by reorganizing the schema by ObjectStore's *schema evolution* mechanism. For example, the corresponding self-organizing process of the schema shown in Fig. 2 is carried out by the following evolution processes:

1. SedanEngine, SportsCarEngine, and BusEngine: inherit Engine.
2. SportsCarEngine overrides attributes: power = High and C.C. = ~Low.
3. SedanEngine overrides attributes: power = ~High and C.C. = ~High.
4. BusEngine overrides attributes: power = High and C.C. = High.
5. Sedan, SportsCar, and Bus: inherit Car.
6. SportsCar overrides the attribute: body = Streamline.
7. Sedan and Bus override the attribute: body = ~Streamline.

5. Developing applications on SLOODS

The application based on SLOODS begins with constructing a database, i.e., the design of schema. The user can design the schema based on IKDM's format with DDL, or construct the schema through GUI with the assist from SLE. After constructing the schema, the schema file is processed by *Schema Generator* (*S-Gen*) which elicits knowledge from schema and adds into AKDM by creating KON for the application. (The design and implementation of S-Gen and parser in Fig. 11. are beyond the scope of our discussion in this study, thus we skip them in sections 4, 5.) Then, the ObjectStore C++ compatible program is generated by S-Gen, compiled by Virtual C++ compiler, and linked with ObjectStore's Class Library. In the same way, The DML, i.e., queries on the schema are optimized by IQE and translated into C++ program by S-Gen. The process is shown in Fig. 11. (Note: The query is optimized at compile-time, rather than run-time. Since ObjectStore only provides compile-environment instead of interpreter-environment, and SLOODS is just it's applications.)
After constructing the schema and database, users can assign/trigger the learning task from GUI. Building an expert system application in SLOODS is also simple by assigning learning tasks for the knowledge acquisition or inserting knowledge into AKDM.

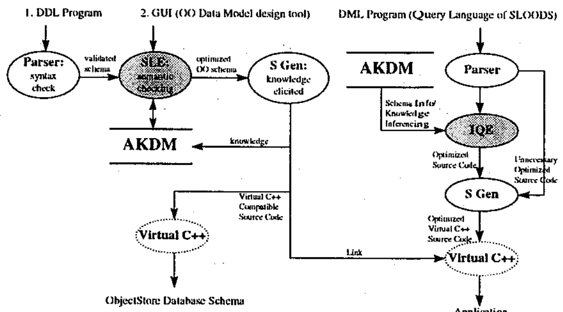
1. DDL Program
2. GUI (OO Data Model design tool)
DML Program (Query Language of SLOODS)
Parser: syntax check
validated schema
SLE: semantic checking
optimized OO schema
S Gen: knowledge elicited
AKDM
knowledge
Virtual C++ Compatible Source Code
Virtual C++
ObjectStore Database Schema
AKDM
Parser
Schema Info/ Knowledge Inferencing
IQE
Optimized Source Code
Unnecessary Optimized Source Code
S Gen
Optimized Virtual C++ Source Code
Link
Virtual C++
Application
Using DDL in SLOODS
Using DML in SLOODS

Fig. 11. Develop an intelligent database application on SLOODS.

6. Conclusion and future work

In this paper, we described a system, SLOODS, to support the development of DBS and KBS applications by mutually taking asset from both systems. The design of AKDM combines the knowledge and the inference mechanism with Rete's efficiency in KON. It also supports dynamic modification of knowledge objects and the self-organization of schema to enhance the semantic of DB schema. ILE, with efficient learning AGE, induces knowledge from the database on behalf of the process of knowledge acquisition, intelligent query optimization, and self-organizing schema.

Database mining is one of the most promising fields for the integration of AI and database systems. For the field, Learning Method and Knowledge Representation are the kernel techniques. In this study, we investigate both fields and develop an integrated intelligent information system. In the future, there are several facets can be enhanced the system such as porting the system to a distributed computing environment, or applying fuzzy concept in KON. The former can investigate the performance of concurrently or parallel inference of KON which is known in the filed of DAI (Distributed AI). As for the later research, currently, the fuzzy concept was applied in ILE only. The FKON (Fuzzy KON) is an interesting study which covers the filed of fuzzy object model. How to integrate the forward/backward inference in KON (or FKON) efficiently is also an interesting area to us.

Reference

[1] J. R, Kent, R. C. Thomas, and I. S. Torsum, "Concept Classifier for Knowledge Acquisition in Frame-Based Systems", in *Knowledge Base Systems*, Vol. 1, No 5, Dec., 1988.

[2] Qing Li and Dennis McLeod, "Conceptual Database Evolution through Learning", in *Object-Oriented Databases with Applications to CASE, Networks, and VLSI CAD*, edited by Rajiv Gupta and Ellis Horowitz, Prentice Hall, 1991.

[3] C. L. Forgy and S. J. Shepard, "Rete: A Fast Match Algorithm", in *AI Expert*, pp. 34-40, Jan., 1987.

[4] Y. M. Huang and S. H. Lin, "An Efficient Learning Method for Object-Oriented Database Using Attribute Entropy" , accepted by *IEEE Trans. on Knowledge and Data Eng.*, Dec., 1996.

[5] S. H. Lin, Y. M. Huang, and Y. S. Duan, "The Design of a Learning Algorithm for Object Oriented Data Model" , *National Computer Symposium*, 1993, Taiwan.

[6] *ObjectStore Reference Manual.*

[7] Shashi Shekhar et al., "Learning Transformation Rules for Semantic Query Optimization: A Data-Driven Approach", in *IEEE Trans. on Knowledge and Data Eng.*, Vol. 5, No. 6, Dec. 1993, pp. 950-964.

[8] Herve Gallaire et al. "Logic and Database : A Deductive Approach", in *ACM Computing Surveys*, Vol. 16, No 2, June 1984.

[9] J. Han, Y. Cai, and N. Cercone, "Knowledge discovery in databases: An attribute-oriented approach", in *Proc. 18th VLDB Conf.*, 1992, pp. 547-559.

[10] S. T. Shenoy and Z. M. Ozsoyoglu, "Design and Implementation of a Semantic Query Optimizer", in *IEEE Trans. on Knowledge and Data Eng.*, Vol. 1, No. 3, Sep. 1989, pp. 344-361.

[11] Michael J. Carey and David J. DeWitt "Extensible Database Systems", in *On Knowledge Base Management Systems*, edited by Michael L. Brodie and John Mylopoulos, Springer-Verlag, 1986.

[12] Michael Stonebraker et. al. "The Design of POSTGRES", in *ACM SIGMOD*, 1986.

[13] Christoph F. Eick and Makolm Taylor, "Integrating Sets, Rules, and Data in an Object-Oriented Environment", in *IEEE Expert*, Feb. 1993.

[14] C. J.. Matheus, P. K. Chan, and G. P. Piatetsky-Shapiro, "Systems for Knowledge Discovery in Databases", in *IEEE Trans. on Knowledge and Data Eng.*, Vol. 5, NO. 6, Dec., 1993.

[15] James Rumbaugh et. al. "*Object-Oriented Modeling and Design*", Prentice-Hall, 1991.

[16] Mark S. Fox and John McDermott "The Role of Databases in Knowledge-Based Systems", in *On Knowledge Base Management Systems*, edited by Michael L. Brodie and John Mylopoulos, Springer-Verlag, 1986.

[17] H. H. Pang, H. J. Lu and B. C. Ooi, "An Efficient Semantic Query Optimization Algorithm", in *Proc. of Seventh Intl. Conf. on Data Eng.*, 1991, pp. 326-333.

[18] R. Elmasri and S. B. Navathe, "*Fundamentals of Database Systems*", Benjamin/Cummings Publishing Company Inc., 1989.

[19] J. R. Quinlan, "Induction of Decision Trees", in *Machine Learning*, Vol. 1,1986, pp. 81-106.

[20] E. A. Rundensteiner, "The Role of AI in Databases Versus the Role of Database Theory in AI: An Opinion", in *Artificial Intelligence in Database System and Information Systems*, North-Holland, IFIP, 1990.

[21] Jay Banerjee and Won Kim, "Semantics and Implementation of Schema Evolution in Object-Oriented Databases", in *ACM SIGMOD*, 1987.

[22] Y. J. Fu and J. W. Han, "DBLearn: A System Prototype for knowledge discovery in relational databases", in *ACM-SIGMOD*, 1994.

[23] *ObjectStore User Guide: LI.*

[24] P. Choen and E. A. Feigenbaum, "Learning and Inductive Inference", in *The Handbook of Artificial Intelligence (Vol. III)*, Chapter XIV, Heuristic Press and William Kaufman, 1983.