TR-0845

Specific Features of Deductive Object-Oriented Database Language QUIXOTE

by
K. Yokota, H. Tsuda & Y. Morita (Oki)

May, 1993

ICOT
Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan
(03)3456-3191 ~ 5

Institute for New Generation Computer Technology

Specific Features of a Deductive Object-Oriented Database Language *Quixote*

Kazumasa Yokota Hiroshi Tsuda Yukihiro Morita *

Institute for New Generation Computer Technology (ICOT)
Mita-Kokusai Bldg. 21F., 1-4-28, Mita, Minato-ku, Tokyo 108, Japan
e-mail: {kyokota,tsuda}@icot.or.jp

Abstract

A deductive object-oriented database (DOOD) language, *Quixote*, has been designed and developed for knowledge information processing applications such as natural language processing, legal reasoning, and genetic information processing in the Japanese Fifth Generation Computer System (FGCS) project and the Follow-On project. As knowledge information is different from conventional information, a new framework for database and knowledge-base management systems was required under such an environment. As a result, *Quixote* has many unique features as a DOOD language and system.

Focusing on such aspects in this paper, we describe specific features of *Quixote* and discuss this approach. Further, we observe very briefly how such features are effective for knowledge information processing by showing some examples of our applications.

1 Introduction

The Japanese Fifth Generation Computer System (FGCS) project developed various hardware and software for knowledge information processing systems under logic and parallelism paradigms over the course of 11 years, since 1982 [10]. The Follow-On project, started in 1993, is intended to carry these objectives forward [21]. During the FGCS project, a new framework of database and knowledge-base management systems was required for representing and managing various kinds of data and knowledge. Under such requirements, *Quixote* was designed and has been, consequently, developed [26]. *Quixote* is a logic programming language with object orientation and database features and is considered a deductive object oriented database (DOOD) [25, 9, 5] language. The main specific characteristics of *Quixote* are as follows:

1) Integration of logic and object-orientation features by *subsumption constraints*.
2) Knowledge classification and inheritance by a *module* concept.
3) Query processing for *partial information databases*: abduction of subsumption constraints and hypothetical reasoning.
4) Support of *database management system features* such as concurrency control, nested transaction, and persistence.

After many efforts to build knowledge representation languages and extensions of deductive databases in the FGCS project, we started to design the *Quixote* language in 1990 and implemented the system as a server in a parallel logic programming language, KL1, on parallel inference machines, PIMs, and as a client in C on UNIX machines. The *Quixote* system has been released as *ICOT Free Software* (the second version in September, 1992, and the third version in June, 1993). We have developed applications such as legal reasoning, genetic information processing, and natural language processing in *Quixote*, and have shown the effectiveness and efficiency of the language. In the Follow-On project, we are now developing new versions of *Quixote* all modules of which will work under UNIX environments, and are developing more knowledge information processing applications.

In this paper, we discuss our approach and describe certain basic and specific features of the *Quixote* language only because of the limited space. For details and

*Knowledge Information Systems Research Lab., Oki Electric Industry Co. Ltd. e-mail: morita@okilab.oki.co.jp

related applications, see other papers [26, 23, 14, 24, 16, 17, 27, 18, 19, 22].

In Section 2, we discuss what is necessary for our environments and why we take this approach in *Quixote*. Next, we briefly explain the concept of an object in *Quixote*: its object identity, subsumption relations and constraints, properties, property inheritance, and methods in Section 3, and its module, database, and query processing in Section 4. In Section 5, we introduce three applications very briefly and show how effective the features of *Quixote* work for them. In the last section, we describe several future plans.

2 Motivation and Policies

2.1 Why DOOD?

When considering 'real' data-intensive applications, we are frequently confronted with the problem of how to represent complex data such as complex objects and partial information, as a predicate in Prolog. To cope with the problem, new constructors, such as tuples and sets, and their corresponding functions are introduced into logic programming as in *LDL* [28] and COL [1]. Also in our environment, we developed a nested relational database management system, *Kappa*, a corresponding logic programming language, *CRL*, and a partial information-based logic programming language, *CIL*, for such applications [26].

However, our attempts to represent more kinds of data and knowledge means the above extensions are half-finished, that is, they are not effective for representing partial information that is frequently used in knowledge information processing. Even if data and knowledge are not necessarily clearly abstracted and formatted, it is important to represent and process them as though they are. For example, discourse understanding is a kind of ambiguity solving in a set of utterances. Processing is expected to correspond to query processing of partial information databases where utterances and background knowledge are stored. From the viewpoint of classifying representations, the above extensions correspond only to value-based representation but not to identity-based representation, which is very useful for representing partial information. As the identity of partial information is naturally treated in the context of object-orientation, we can expect logic and object-orientation to be integrated in our environment.

From the viewpoint of data modeling, object-orientation concepts are important not only for identity-based representation but also for encapsulation, inheritance, and method. For example, encapsulation and method are key concepts for a protocol of cooperative problem solving over heterogeneous distributed knowledge-bases, and inheritance is essential for classification of knowledge.

From another point of view, a DOOD is not only a problem in a database but also in other areas, such as natural language processing, artificial intelligence, and programming languages. For example, deductive databases are related to programming languages and artificial intelligence, and complex objects are related to feature structures in natural language processing and record structures in programming languages. In knowledge information processing especially, many language concepts, such as knowledge representation, programming, situated inference, knowledge-bases, and databases are closely related. A DOOD approach as the integration of logic and object-orientation paradigms is not only appropriate for such environments but is also expected to be a key concept for integration of more concepts.

2.2 Design Policies for *Quixote*

A DOOD approach is not only related to extensions of data structure or representation but also to logical and computational modeling extensions, because they are mutually related. The problem is in knowing what forms the base of the integration of logic and object-orientation paradigms. Here, we consider the following criteria:

1) Are all properties of an object homogeneous?
2) How is an object identity represented?
3) How is extrinsic properties represented?
4) Which semantics should be selected for sets and how are they treated?
5) How should globality and locality of data and knowledge be introduced?
6) What extensions are needed for query processing?
7) What about relationship to traditional object-orientation concepts?
8) What about relationship to deductive databases?
9) What about relationship to database management features?

10) Where should a new language be placed on the whole?

The *first* criterion concerns the classification of properties, as there are intrinsic and extrinsic properties for any object. The former contributes to discrimination of objects, that is, corresponds to a constituent of value-based representation, while the latter is incidental for an object, that is, the update does not change the existence of the object itself. For example, two phrases, "$\cdots$ *apple which is red* $\cdots$" and "$\cdots$ *apple, which is red,* $\cdots$" are different, that is, although *which is red* is common, the former is restricted reading, that is, intrinsic, while the latter is non-restricted reading, that is, incidental or extrinsic, where *apple* plays the role of identity. Representation of an object with both properties corresponds to integration of value-based and identity-based representations.

The *second* criterion concerns the representation of an object identity, i.e., object identifier (oid). Traditionally, an oid is represented in the form of a pointer or an address and is hidden from users in most object-oriented languages, while a name, corresponding to an oid, is explicitly used by users. We take a term consisting of intrinsic properties as a name, which also can be an oid for users. If users want to avoid the tedious work of naming all objects, it seems to be rather easy to attach an oid generation mechanism as in ILOC [6]. As the correspondence of a name and a concept is useful, it is also desirable to use a name to represent self-referential concepts such as 'a person who employs himself'. To cope with such problems, the infinite structure should be embedded in an oid or a name. Thus, we do not take a predicate-based notation as an id-term in F logic [8], but tuple-based representation with tag such as $X{:}person[employ \Rightarrow X]$, which is similar to ψ term [3].

The *third* criterion concerns representation of extrinsic properties and the relation between oids. The representation of partial information should be flexible. Given an oid o, a label l, and a value v, an extrinsic property is represented as a constraint between $o.l$ (l-value of o) and v. The constraint is closely related to the relation between oids, because properties should be inherited between objects according to the relation. We take the *subsumption* (a kind of ISA) relation as the relation, consider a constraint solver over a set of subsumption constraints, and take the idea in DOT [20] as the inheritance mechanism. From the requirements of termination and confluency of a constraint solver, CLP(AFA) [15] is known to have a constraint domain of hypersets or non-well-founded sets [2], which are appropriate for representing infinite structures. Thus, we decided to extend this solver over constraints between sets, because an extrinsic property may take a set as its value. As the result, the semantics of an oid and a label of an extrinsic property is defined as a hyperset and a function over hypersets, respectively.

The *fourth* criterion concerns how to treat sets, that is, ordering of sets, set grouping or set construction, and representation of complex objects.

1) Some kinds of orderings among sets where elements are ordered are known, as in [4]: Hoare ordering, Smyth ordering, and Egli-Milner ordering, as well as ordinal inclusion ordering. As the introduction of all orderings makes the language too complex, we selected Hoare ordering, because it seems to be natural in most of our applications when considering the subsumption relation. We have a plan to design a meta-language where other orderings are also definable.

2) Set operations such as set grouping are very useful and powerful but the semantics is of a higher order. So we take an alternative, that is, as the least upper bound of sets under Hoare ordering corresponds to a set union, we realize set grouping by combining Hoare ordering and property inheritance.

3) Set constructors are introduced into subsumption constraints and make it possible to represent complex objects. However, as intrinsic properties with sets seem to cause some semantic problems, we take an ϵ-term approach [3].

The *fifth* criterion concerns the globality and locality of data and knowledge in a database, which also relate to the construction of very large knowledge-bases. To realize the coexistence of inconsistent knowledge or localization of properties or methods, we introduce a *module* concept as in [11, 12, 13]. A module can encapsulate inconsistent properties or methods locally. Although the problem is whether a module is an object or not, we do not take a module as an object to avoid higher-order semantics. Alternatively, a module identifier (mid) is in the form of a term, which realizes a parametric module and propagated constraints globally among modules.

The *sixth* criterion concerns extensions of query processing.

1) Knowledge information processing applications require various reasoning such as deduction, abduction, and induction, as well as domain-dependent heuristics. Focusing on the partiality of information, we introduce (restricted) abduction (or hypothesis generation) and hypothetical reasoning. Such features are very useful for treating databases as thinking experimental tools, which are frequently used in knowledge information processing environments.

2) Such applications use not only 'high-level' data such as rules but also 'low-level' data such as DNA sequence. The latter needs information retrieval features such as homology searching. As it is difficult to embedded a large number of functions into one language, we decided to consider such features as cooperative problem solving in a heterogeneous distributed environment.

The *seventh* criterion concerns the relationship to traditional object-orientation concepts.

1) Is it necessary to classify class and instance as in traditional object-orientation? By taking a term as an oid, a term with variables can be considered to be a class over the instantiated ones. As its classification hierarchy seems to be more general, we decided not to discriminate class from instance but to employ relative classification by using terms with variables.

2) Is it necessary to introduce types? Although a type system is necessary in heterogeneous (distributed) environments, subsumption constraints can be an alternative in an homogeneous environment, so, we decided to postpone the introduction of a type system.

3) How is a method represented? As a property is considered to be a method as in F-logic [8], a label should be in the form of a term.

4) How does an object work autonomously? As asynchronous communication features should be embedded into each object to make it fully autonomous, we postpone to embed such features in the current specification, because most *object-oriented* databases and languages have ignored them.

The *eighth* criterion concerns the relationship with deductive databases. Although negation and disjunction have been well studied in logic programming and deductive databases, subsumption constraints with negation or disjunction have not been studied. Thus, we have not introduced such logical components in the earlier versions. Further, as bottom-up evaluation is not known in constraint logic programming except with finite domains, we employ top-down evaluation and pursue optimization. To retain upwards compatibility with deductive databases, we decided also to employ ordinary predicate notations as terms in *Quixote*.

The *ninth* criterion concerns the relationship to database management features. The problem is deciding which features of database management systems should be supported in a language.

1) In requests to extensions of query processing, update and nested transactions should be supported.

2) For persistence, we designed a logical interface to external persistence store such as UNIX file systems and external database management systems. Persistence is given to all objects except temporal ones generated for query processing.

3) Although integrity constraint and concurrency control are essential in a database, we decided to implement only minimal features in the earlier versions.

The *tenth* criterion concerns where we place the language on the whole. We discussed whether *Quixote* might be quixotic or not (the name was derived differently), because the language must have many features. We consider this integration to be rather moderate but not an unreasonable integration. As a result, *Quixote* has many aspects: a DOOD language, a knowledge representation language, a constraint logic programming language, a database programming language, and a situated programming language.

Although we had criteria besides the above (parallel processing, programming environment, and system architecture), we do not describe them here because of limited space. Please see the details in [17].

3 Basic Features of *Quixote* Objects

An object in *Quixote* consists of an object identifier (oid) and a set of properties. Each property can be considered as a method as usual and the implementation of a method is written in the body of a rule. The

subsumption relation among oids makes property inheritance possible.

3.1 Object Identity and Subsumption Constraints

An oid is in the form of a tuple called an *object term*. For example,

$$apple.$$

$$apple[color = red], \text{ and}$$

$$cider[alcohol = yes, \\ product = process[source = apple, \\ process = ferment]]$$

are object terms, where *apple* is *basic*, but the latter two are *complex*. Although an infinite structure and set constructors are introduced, as explained in the second and fourth criteria in Section 2.2, we do not explain them here for simplicity.

Given *subsumption relation* (partial order) $\sqsubseteq$ among basic object terms, the relation is extended among complex object terms as usual. For example,

$$apple \sqsupseteq apple[color = red].$$

Congruence relation $o_1 \cong o_2$ is defined as $o_1 \sqsubseteq o_2 \ \wedge \ o_1 \sqsupseteq o_2$. We assume that the subsumption relation among object terms constitutes a lattice without loss of generality because it is easy to construct a lattice from a partially ordered set, as in [3]. Meet and join operations of object terms are denoted by $\downarrow$ and $\uparrow$, respectively.

Properties are defined as a set of subsumption constraints of an oid and used with the oid as follows:

$$apple|\{apple.species \sqsubseteq rose, \\ apple.area \sqsupseteq_H \{aomori, nagano\}\}.$$

where *apple* is an object term and the right hand side of $|$ is a set of properties: $apple.species \sqsubseteq rose$ means that *apple*'s *species* is (subsumed by) *rose* and $apple.area \sqsupseteq_H \{aomori, nagano\}$ means that there are *aomori* and *nagano* in *apple*'s production *areas*. Here a relation between sets is defined as Hoare ordering based on subsumption relation:

$$S_1 \sqsubseteq_H S_2 \stackrel{def}{=} \forall e_1 \in S_1. \exists e_2 \in S_2. \ e_1 \sqsubseteq e_2.$$

Although Hoare ordering is not partial, we assume it as a partial order because the representative of an equivalence class modulo $\sqsubseteq_H$ is easily defined as a set where any element is not subsumed by other elements in the same set.

3.2 Property Inheritance

For *property inheritance*, we assume the following rule as in [20]:

> if $o_1 \sqsubseteq o_2$,
> then $o_1.l \sqsubseteq o_2.l$ and $o_1.l' \sqsubseteq_H o_2.l'$.

where o_1 and o_2 are object terms, l and l' are labels, and l and l' take a single value and a set value, respectively. According to the rule, we get, for example,

> if $apple.species \sqsubseteq rose$,
> then $apple[color = red].species \sqsubseteq rose$.
> and
> if $apple[color = red].area \sqsupseteq_H \{fukushima\}$,
> then $apple.area \sqsupseteq_H \{fukushima\}$.

That is, $apple.species \sqsubseteq rose$ is downward inherited from *apple* to $apple[color = red]$, while $apple[color = red].area \sqsupseteq_S \{fukushima\}$ is upward inherited from $apple[color = red]$ to *apple*.

Note that there are two kinds of properties: properties in an object term and properties in the form of constraints. The former are called *intrinsic* and the latter are called *extrinsic*. Only extrinsic properties (subsumption constraints) are inherited according to the (extended) subsumption relation among object terms.

Intrinsic properties interrupt property inheritance as follows:

> Even if *apple* has $apple.color \cong green$,
> $apple[color - red]$ does not inherit $color \sqsubseteq green$
> because the intrinsic property $color = red$
> with the same label *color*
> rejects the extrinsic property $color \sqsubseteq green$.

This corresponds to *exception* of property inheritance.

Multiple inheritance is defined as the merging of subsumption constraints. Such constraints are reduced as follows:

> if $p.l \sqsubseteq a$ and $o.l \sqsubseteq b$, then $o.l \sqsubseteq a \downarrow b$,
> if $a \sqsubseteq o.l$ and $b \sqsubseteq o.l$, then $a \uparrow b \sqsubseteq o.l$,
> if $o.l \sqsubseteq_H s_1$ and $o.l \sqsubseteq_H s_2$, then $o.l \sqsubseteq_H s_1 \cup s_2$,
> and
> if $s_1 \sqsubseteq_H o.l$ and $s_2 \sqsubseteq_H o.l$,
> then $\{x \downarrow y | x \in s_1, y \in s_2\} \sqsubseteq_H o.l$.

Note that the least upper bound of the two sets, s_1 and s_2, is defined as $s_1 \cup s_2$ under Hoare ordering, because $s_1 \cup s_2 \ \sqsubseteq_H \ \{e_1 \uparrow e_2 | \ e_1 \in s_1, e_2 \in s_2\}$. In the above example, the merging of $apple.area \sqsupseteq_H \{aomori, nagano\}$

and $apple.area \sqsupset_H \{fukushima\}$ is reduced to $apple.area \sqsupset_H \{nanuri.nagano.fukushima\}$.

3.3 Intensional Objects

An object can be defined intensionally in the form of a *rule*:

$$o_0|C_0 \Leftarrow o_1|C_1, \cdots, o_n|C_n \parallel C.$$

where, for $0 \leq i \leq n$, o_i is an object term and C_i is a set of the related subsumption constraints, and C is a set of constraints (variable constraints). An object $o_0|C_0$ is intensionally or conditionally defined by the rule. $o_0|C_0$ is a *head* and $o_1|C_1, \cdots, o_n|C_n \parallel C$ is a *body*. Intuitively, a rule means that if the body is satisfied then the head is satisfied. If a body is empty, then the rule is called a *fact*. In a sense, an object is defined as a set of rules with the same object term. There is one important restriction in C_0: C_0 may not contain subsumption relations among object terms. The reason for this is to avoid destruction of a lattice during query processing.

Note that an object term plays the role of an oid. That is, two facts,

$$o[\{o.l \sqsubseteq o\} \Leftarrow \text{ and}$$
$$o[\{o.l \sqsubseteq b\} \Leftarrow$$

can be merged as follows:

$$o[\{o.l \sqsubseteq a \downarrow b\} \Leftarrow .$$

In cases where an object term with variables is in the head of a rule, an object is defined when the variables are instantiated during query processing. That is, subsumption constraints in a head are merged after evaluation of all the related rules.

When the constraints of a head are empty, QUIXOTE is an instance of CLP(X) [7]:

$$o_0 \Leftarrow o_1|C_1, \cdots, o_n|C_n \parallel C$$
$$\iff \quad o_0 \leftarrow o_1, \cdots, o_n \parallel C_1 \cup \cdots \cup C_n \cup C$$

From a programming language point of view, the existence of head constraints in a rule makes QUIXOTE an extension of CLP(X) and, in the procedural semantics, the possibility of merging head constraints must be checked at every OR node of the resolution tree. See the details in [16].

4 Databases and Query Processing

4.1 Modules and Databases

A set of rules can be defined as a *module*:

$$m :: \{r_1, \cdots, r_n\}.$$

This means that a module identified by a *module identifier* (mid) m has rules $r_1, \cdots, r_n$. We use 'module m' instead of 'a module identified by mid m' for simplicity. Here, we define the *submodule relation* between modules. For example, consider two submodule relations:

$$m_1 \sqsupseteq_S m_2 + m_3, \text{ and}$$
$$m_2 \sqsupseteq_S m_4.$$

The definitions mean that m_1 inherits all rules in m_2 and m_3, and m_2 inherits all rules in m_4. We call such inheritance *rule inheritance*, where exception, locality, and overriding are also defined [24]. The submodule relation is an acyclic directed graph, in which modules can be nested. Rules without a mid are inherited by all modules. For example, consider the following three definitions:

(1)
$$m_1 :: \{r_{11}, \cdots, r_{1n}\}$$
$$m_2 :: \{r_{21}, \cdots, r_{2m}\}$$
$$\{m_1, m_2\} :: \{r_{31}, \cdots, r_{3l}\}$$

(2)
$$m_1 :: \{r_{11}, \cdots, r_{1n}\}$$
$$m_2 :: \{r_{21}, \cdots, r_{2m}\}$$
$$r_{31}, \cdots, r_{3l}$$

(3)
$$m_1 :: \{r_{11}, \cdots, r_{1n}\}$$
$$m_2 :: \{r_{21}, \cdots, r_{2m}\}$$
$$common :: \{r_{31}, \cdots, r_{3l}\}$$
$$m_1 \sqsupseteq_S common$$
$$m_2 \sqsupseteq_S common$$

In any of these definitions, m_1 has $r_{11}, \cdots, r_{1n}, r_{31}, \cdots, r_{3l}$ and m_2 has $r_{21}, \cdots, r_{2m}, r_{31}, \cdots, r_{3l}$.

A module can be referenced from a subgoal in a rule in other modules. The definition of a rule is extended as follows:

$$m_0 :: o_0|C_0 \Leftarrow m_1 : o_1|C_1, \cdots, m_n : o_n|C_n \parallel C$$

which means that there is a rule in module m_0 such that if $o_i|C_i$ and C are satisfiable in module m_i for all $1 \leq i \leq n$, then $o_0|C_0$ is satisfiable in a module m_0. There might be some discussion about why a rule is not defined as follows:

$$m :: m_0 : o_0|C_0 \Leftarrow m_1 : o_1|C_1, \cdots, m_n : o_n|C_n \parallel C.$$

According to the definition, module m knows some knowledge in another module, that is, the rule corresponds to a kind of *brief*. However, it causes serious semantical problems.

Although the module is introduced as the fifth criterion in Section 2.2, we can list three objectives:

1) Classification of data and knowledge under certain criteria.
2) Coexistence of inconsistent data and knowledge.
3) Introduction of a modular programming style.

These features are also very useful for constructing very large knowledge-bases (VLKB).

A *QUIXOTE database* or a *QUIXOTE program* is defined as a triple (S, M, R) of a set S of subsumption relations among basic objects, a set M of submodule relations among mids, and a set R of rules. Intuitively, a database can be also considered as a set of modules or as a set of objects.

4.2 Query Processing

One of the major characteristics in knowledge information is the partiality of information, that is, sufficient information is not necessarily given, such as in business applications. The introduction of an object identity is essential for representing such partial information. Partiality should be considered not only in representation but also in query processing:

1) What information is lacking in the database for this query?
2) If some information is inserted into the database, what answer will be gotten for this query?

Further, as derivation becomes more complicated, we want to know why a particular answer is returned. As details of such query processing are presented in [27], we outline the processing here.

In logic programming, finding a lack of information or unsatisfiable subgoals corresponds to abduction, that is, hypothesis generation. Remember that a rule in QUIXOTE can be represented as follows:

$$o_0|C_0 \Leftarrow o_1, \cdots, o_n \,||\, C_1 \cup \cdots \cup C_n \cup C,$$

where oids $o_1, \cdots, o_n$ are considered as existence checks of the corresponding objects, while $C_1 \cup \cdots \cup C_n \cup C$ is considered to be a satisfiability check of subsumption and variable constraints. In the current implementation of QUIXOTE, only subsumption constraints in $C_1 \cup \cdots \cup C_n \cup C$ are taken as assumptions, that is, even if body constraints are not satisfied, they are taken as a conditional part of an answer. For example, consider a database consisting of three objects:

$$o_1 \Leftarrow o_2||\{o_2.l \sqsubseteq a\}.$$
$$o_2.$$
$$o_3 \Leftarrow o_4.$$

For a query ?-o_1, the answer is that if $o_2.l \sqsubseteq a$, then yes, while, for a query ?-o_3, the answer is no.

Further, the derivation process of an answer is also returned as an explanation with the answer. That is, each answer is in the following form:

if *assumptions* then *answer* because *explanation*,

where both *assumptions* and *answer* are in the form of a set of constraints.

On the other hand, hypothetical reasoning corresponds to the insertion of hypotheses into a database. A query is in the following form:

if *hypotheses* then ?-*query*
(written as ?-*query*::*hypotheses*).

For example, consider the database used above. For queries ?-$o_1 :: o_2|\{o_2.l \sqsubseteq a\}$ and ?-$o_3 :: o_4$, both answers are yes without any assumptions. That is, if, for a query ?-q, the answer is "if H then A", then, for a query ?-$q :: H$, the answer is simply A. Note that, as a database consists of a triple of definitions of subsumption relations, submodule relations, and rules, hypotheses can also consist of such a triple. A query ?-$q :: (S_H, M_H, R_H)$ to a database (S, M, R) is equivalent to a query ?-q to a database $(S \cup S_H, M \cup M_H, R \cup R_H)$.

Hypotheses are incrementally inserted into a database, that is, a query ?-$q :: H$ to a database DB updates the database to $DB \cup H$. To control such repetitive insertions of hypotheses, nested transactions are introduced. Users can declare *begin_transaction*, *abort_transaction*, or *end_transaction* at any time among queries. A top-level commit operation (an outermost transaction from *begin_transaction* to *end_transaction*) makes insertions persistent. On the other hand, a *roll-back* operation (caused by *abort_transaction*) recovers the before image of the corresponding *begin_transaction*. Hypothetical reasoning is useful in the construction of a knowledge-base or in *thinking experiment* or *trial-and-error* type query processing.

4.3 Other Features

Here, we list some more features of QUIXOTE:

1) *Assertion* and *deletion* of extensional objects and properties during query processing are supported as in [28]. These are controlled by the same uniform nested transaction logic used in hypothetical reasoning. However, note that the subsumption relation and submodule relation may not be updated during query processing, because the change in their inheritance might destroy the soundness of the derivation, although they may be inserted as hypotheses.

2) All objects in QUIXOTE except temporarily created objects during query processing are persistent. Persistent objects in a database are stored through a uniform logical interface into other database management systems or file systems via TCP/IP protocol. Such objects are invoked when the corresponding database is opened.

3) A QUIXOTE system consists of a client as a user interface and a server as a knowledge-base engine. Servers and clients are connected also by TCP/IP protocol and servers control multi-user access.

5 Applications

We have developed several applications mainly in QUIXOTE, which are rather new for database communities. Details are reported in other papers [22, 18, 19, 26].

5.1 Legal Reasoning

Our legal reasoning system [22, 26] aims at the prediction of judgments for given new cases. To meet this objective, many databases have been written in QUIXOTE: for example, statutes, theories, precedents, and a legal concept dictionary. Each of these corresponds to a module in QUIXOTE. The important point of the prediction is to find similar precedents with a given new case and to construct similar legal reasoning for the new case.

The features of QUIXOTE closely relate to the process and work effectively and efficiently:

1) To find similar precedents, the constraints embedded in the precedents and a new case are gradually relaxed according to subsumption hierarchy. For example, if $a \sqsubseteq b$, then $X \sqsubseteq a$ can be relaxed to $X \sqsubseteq b$. As the control of relaxation is not a feature of QUIXOTE, it is written by the user.

2) As connection among modules is dynamic, hypothetical reasoning in QUIXOTE is essential. For example, users try to connect or disconnect various modules to improve judgment.

3) As precedents are generally incomplete descriptions, abduction in QUIXOTE plays an important role in the process. For example, if we could find one new fact in some precedent, the precedent might become similar to a new case.

4) Explanation of an answer is also indispensable to the verification of derived judgments. For example, users want to know which statutes, precedents, or theories are used for some judgment.

A legal reasoning system, which needs a large number of databases and knowledge-bases, is a typical application in artificial intelligence and seems to be good for database communities too.

5.2 Genetic Information Processing

There are various kinds of data and knowledge in molecular biology, such as sequence and structure data of genes and proteins, maps of sequences, and metabolic reactions. Such data is stored in two kinds of databases: public (text-based) databases such as GenBank, PDB, and ProSite, and biologists' private databases. There is frequently duplication and inconsistencies between these databases. The problem in our environment is to provide a framework for an integrated 'inconsistent' database with various kinds of data and knowledge, which help biologists' experiments, and to build such an integrated experimental database. However, as considered in the sixth criterion in Section 2.2, all data and knowledge cannot be stored in QUIXOTE. So, we recognize two kinds of data: low level data to be stored in a nested relational database, Kappa, and high-level data to be written in QUIXOTE, although users must be responsible for their integration into the current implementation of Kappa and QUIXOTE.

We describe three examples:

1) The first example is a description of protein functions and motifs. For example, consider a chemical reaction between enzymes and co-enzymes, whose

scheme is as follows:

$$\text{Sources} + \text{Co-enzymes} \xrightarrow[\text{Environments}]{\text{Enzymes}} \text{Products}$$

Such reactions correspond to rules and the chains correspond to transitive rules. *Quixote* is particularly appropriate for representing structured combinations and their inferences.

2) The second example is a description of experimental data. For example [18], the knowledge item *cytochromes have a certain feature* is sometimes reconsidered as follows:

   - *cytochromes and hemoglobins have a certain feature*, or
   - *cytochrome c has a certain feature*.

   As most erroneous identifiers change their abstraction level, an oid and subsumption relation in *Quixote* is appropriate for representing such objects.

3) Experimental data are sometimes inconsistent. For example, the multiple alignment of sequences and prediction of protein structure generate many hypotheses which are not verified. A module in *Quixote* is appropriate for storing and classifying such data.

One problem in molecular biological databases is how to integrate *Quixote* databases and ordinary databases.

5.3 Natural Language Processing

In our environment, we are engaged in various problems concerning natural language processing, such as discourse understanding, constraint-based grammar, and situated inference. Of these, we focus on constraint-based grammar and situated inference as applications of *Quixote*. Here, we describe an example of situated inference.

Concepts in situation theory are rephrased in *Quixote* as follows [19, 26]:

infon	$\leftrightarrow$	object term
role	$\leftrightarrow$	label
parameter set	$\leftrightarrow$	object term with subsumption constraints
situation	$\leftrightarrow$	module
support relation	$\leftrightarrow$	membership in module

We described some examples of situated inference in *Quixote*, according to the above correspondence for the following objectives:

1) How to explicate hidden parameters by constraints.

2) How to describe situated inference from perspectives such as tense and aspects.

For example, consider the different knowledge held by two speakers (Quine's example).

A: "If Bizet and Verdi are compatriots, then Bizet is Italian."

B: "If Bizet and Verdi are compatriots, then Verdi is French."

Each speaker has different hidden knowledge, that is, A assumes knowledge such that Verdi is Italian and such knowledge is taken as an assumption in *Quixote* query processing. The example is written in *Quixote* as in Figure 1, where "::" is a delimiter between definitions. Details in the query processing are reported in [27].

$$
\begin{array}{l}
nation \sqsupseteq italy::\\
nation \sqsupseteq france::\\
speaker\_a \sqsupseteq_S common::\\
speaker\_b \sqsupseteq_S common::\\
speaker\_a :: \{bizet | \{bizet.nationality \cong italy\} \Leftarrow\\
\qquad compatriots[per1 = bizet.\\
\qquad\qquad per2 = verdi]\}::\\
speaker\_b :: \{verdi | \{verdi.nationality \cong france\} \Leftarrow\\
\qquad compatriots[per1 = bizet.\\
\qquad\qquad per2 = verdi]\}::\\
common :: \{compatriots[per1 = X.per2 = Y] \Leftarrow\\
\qquad X | \{X.nationality \cong N1\},\\
\qquad Y | \{Y.nationality \cong N2\}\\
\qquad || \{N1 \sqsubset nation.\\
\qquad\quad N2 \sqsubset nation.\\
\qquad\quad N1 \cong N2\}::\\
\qquad bizet::\\
\qquad verdi\}
\end{array}
$$

Figure 1: Qnine's Example in *Quixote*

6 Concluding Remarks

Four features of *Quixote*, explained in Section 1, can be considered to be effective and efficient from var-

ions aspects, as shown in the applications in Section 5. Our contributions to integration of logic and object-orientation paradigms are intended to propose subsumption constraints as key concept and to show their effectiveness in a knowledge information processing environment. Another contribution to knowledge information processing is the formulation of query processing of partial information databases and the demonstration of the usefulness of a module concept in databases and knowledge-bases. The aim of this paper is to discuss this approach and its related problems.

Future planned works includes the following:

1) In the current implementation of *Quixote*, we have not introduced *logically indefinite* knowledge, such as negation and disjunction. However, such logical extensions are strongly required for applications such as legal reasoning. As there are some problems in extending the constraint domain, we will introduce negation and disjunction for existence checks of objects.

2) To control the computation process, the facilities of *meta-programming* are very important, as in Prolog. Especially, such facilities are very effective for introducing heuristics depending on applications, and are also expected to improve the efficiency of computation.

3) In the current implementation of *Quixote*, the discrimination of *locality and globality* is fixed. For example, the subsumption relations are global over modules. However, there are many applications which require local definitions. We consider a new framework incorporating both locality and globality.

4) As discussed in the fourth criterion in Section 2.2, the semantics of some constructs should be definable. Further, users frequently want to change the syntax for their applications. A *platform language* is desirable for such definitions of both syntax and semantics.

5) The universal language for all problem domains is quixotic, as in human languages. As discussed in the sixth criterion in Section 2.2 and in Section 5.2, *cooperative problem solving* in a heterogeneous distributed environment is necessary for many practical applications.

6) Although we have developed some experimental applications, we will develop more *'big'* applications under a knowledge information processing environment.

7) While a client of *Quixote* works in a UNIX environment, the server works only in restricted environments which support KL1. We plan to port the server into an UNIX environment in order to distribute *Quixote* software widely.

Acknowledgment

The authors would like to thank Prof. Kazuhiro Fuchi of the University of Tokyo (former director of ICOT Research Center) and Dr. Shunichi Uchida of the director of ICOT Research Center for their continuous encouragement, and all members of the *Quixote* project for the almost daily discussions we had with them on for the design and development of *Quixote*.

References

[1] S. Abiteboul and S. Grumbach, "COL: A Logic-Based Language for Complex Objects", *Proc. Int. Conf. on Extending Database Technology*, in *LNCS* 303, Springer, 1988.

[2] P. Aczel, *Non-Well Founded Set Theory*, CSLI Lecture notes No. 14, 1988.

[3] H. Aït-Kaci, "An Algebraic Semantics Approach to the Effective Resolution of Type Equations", *Theoretical Computer Science*, no.45, 1986.

[4] P. Buneman and A. Ohori, "Using Powersetdomains to Generalize Relational Databases", *Theoretical Computer Science*, no.91, 1991.

[5] C. Delobel, M. Kifer, and Y. Masunaga (eds.), *Deductive and Object-Oriented Databases. (Proc. 2nd Int. Conf. on Deductive and Object-Oriented Databases. (DOOD'91)*, LNCS 566, Springer, 1991.

[6] R. Hull and M. Yoshikawa, "ILOG: Declarative Creation and Manipulation of Object Identifiers", *Proc. Int. Conf. on Very Large Data Bases*, 1990.

[7] J. Jaffer and J.-L Lassez, "Constraint Logic Programming", *Proc. 4th IEEE. Symp. on Logic Programming*, 1987.

[8] M. Kifer and G. Lausen. "F-Logic A Higher Order Language for Reasoning about Objects, Inheritance, and Schema", *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pp.134-146, Portland, June, 1989.

[9] W. Kim, J.-M. Nicolas, and S. Nishio (eds.), *Deductive and Object-Oriented Databases, (Proc. 1st Int. Conf. on Deductive and Object-Oriented Databases, (DOOD89))*, North-Holland, 1990.

[10] T. Kurozumi, "Overview of the Ten Years of the FGCS Project", *Proc. Int. Conf. on Fifth Generation Computer Systems*, ICOT, Tokyo, June 1-5, 1992.

[11] D. Miller, "A Theory of Modules for Logic Programming", *The Int. Symp. on Logic Programming*, 1986.

[12] L. Monterio and A. Porto, "Contextual Logic Programming", *The Int. Conf. on Logic Programming*, 1989.

[13] L. Monterio and A. Porto, "A Transformational View of Inheritance in Programming", *The Int. Conf. on Logic Programming*, 1990.

[14] Y. Morita, H. Haniuda, and K. Yokota, "Object Identity in *Quixote*", *Proc. SIGDBS and SIGAI of IPSJ*, Oct., 1990.

[15] K. Mukai, "CLP(AFA): Coinductive Semantics of Horn Clauses with Compact Constraint", *The 2nd Conf. on Situation Theory and Its Applications*, Kinloch Rannoch Scotland, Sep., 1990.

[16] T. Nishioka, H. Tsuda, and K. Yokota, "The Procedural Semantics of a Deductive Object-Oriented Database Language *Quixote*", *Proc. Joint Workshop of SIGDBS of IPSJ and SIGDE of IEICE*, July 21-23, 1993.

[17] C. Takahashi, Y. Morita, M. Nishioka, and H. Tsuda, "Features and Implementation of a Deductive Object-Oriented Database System *Quixote*", *Proc. Joint Workshop of SIGDBS of IPSJ and SIGDE of IEICE*, July 21-23, 1993.

[18] H. Tanaka, "Integrated System for Protein Information Processing", *Proc. Int. Conf. on Fifth Generation Computer Systems*, ICOT, Tokyo, June 1-5, 1992.

[19] S. Tojo and H. Yasukawa, "Situated Inference of Temporal Information", *Proc. Int. Conf. on Fifth Generation Computer Systems*, ICOT, Tokyo, June 1-5, 1992.

[20] M. Tsukamoto, S. Nishio, and M. Fujio, "DOT: A Term Representation Using DOT Algebra for Knowledge Representation", in [5], 1991.

[21] S. Uchida, R. Hasegawa, K. Yokota, T. Chikayama, K. Nitta, and A.Aiba, "Outline of the FGCS Follow-on Project", *New Generation Computing*, vol.11, pp. 217-222, 1993.

[22] N. Yamamoto, "TRIAL: A Legal Reasoning System (Extended Abstract)", *Joint French-Japanese Workshop on Logic Programming*, Renne, France, July, 1991.

[23] H. Yasukawa and K. Yokota, "Labeled Graphs as Semantics of Objects", *Proc. SIGDBS and SIGAI of IPSJ*, Oct., 1990.

[24] H. Yasukawa, H. Tsuda, and K. Yokota, "Object, Properties, and Modules in *Quixote*", *Proc. Int. Conf. on Fifth Generation Computer Systems*, ICOT, Tokyo, June 1-5, 1992.

[25] K. Yokota and S. Nishio, "Towards Integration of Deductive Databases and Object-Oriented Databases — A Limited Survey", *Proc. Advanced Database System Symp.*, Kyoto, Dec., 1989.

[26] K. Yokota and H. Yasukawa, "Towards an Integrated Knowledge Base Management System", *Proc. Int. Conf. on Fifth Generation Computer Systems*, ICOT, Tokyo, June 1-5, 1992.

[27] K. Yokota, Y. Morita, H. Tsuda, H. Yasukawa, and S. Tojo, "Query Processing for Partial Information Databases in *Quixote*", *submitted for publication*.

[28] C. Zaniolo, "Design and Implementation of a Logic Based Languages for Data Intensive Applications", *Proc. Int. Conf. and Symp. on Logic Programming*, 1988.