Knowledge Representation Language *QUIXOTE*

Hiroshi Tsuda and Kazumasa Yokota
Institute for New Generation Computer Technology (ICOT)
1-4-28 Mita, Minato-ku, Tokyo 108, Japan
{tsuda,kyokota}@icot.or.jp

Abstract

This paper outlines the language and implementation of the knowledge representation language *QUIXOTE*.

QUIXOTE is a hybrid language of deductive object-oriented database (DOOD) and constraint logic programming (CLP) based on subsumption relations. The new mechanisms of *QUIXOTE* are a combination of object-orientation concepts such as object identity and property inheritance, and the concept of a module that classifies a large knowledge base. In addition, its logical inference system is extended to be able to make hypothetical reasoning and restricted abduction. Such features play important roles in applications such as legal reasoning, biological databases, and natural language understanding.

There are two kinds of *QUIXOTE* implementation; big-*QUIXOTE* which is a full client-server style implementation in KLIC (KL1) and micro-*QUIXOTE* which is small restricted implementation in C.

1 Introduction

QUIXOTE is designed as a hybrid language for deductive object-oriented database (DOOD[9, 6]) and constraint logic programming (CLP) based on subsumption relations. The new mechanisms of *QUIXOTE* are a combination of object-orientation concepts such as object identity and property inheritance, and the concept of a module that classifies a large knowledge base[20, 22, 24]. In addition, its logical inference system is extended to be able to make hypothetical reasoning and restricted abduction. Such features play important roles in applications such as legal reasoning, biological databases, and natural language understanding.

Section 2 describes the *QUIXOTE* language including several new issues such as NAF (negation as failure), disequation constraints, and special built-in modules. Section 3 shows implementation issues for big-*QUIXOTE* and micro-*QUIXOTE*. Section 4 gives a brief overview for *QUIXOTE* application such as legal reasoning, biological information processing, and natural language understanding.

2 *QUIXOTE* language

In this section, we overview the knowledge representation features of *QUIXOTE*.

2.1 Object Term and Subsumption Relation

Simple concepts can be represented as atoms called *basic objects*. We assume a set B of basic objects. For example, the following are basic objects:

$$mozart, person, piano, violin, instrument$$

The set of basic objects are partially ordered by $\preceq$. For example,

$$mozart \preceq person,$$
$$piano \preceq instrument, \quad violin \preceq instrument$$

Relations between concepts such as "Mozart *is a* person," and "piano is *a kind of* instrument" can be represented by this partial ordering. For simplicity, we assume that $\preceq$ is a strict order without circularity.

We represent complex concepts, such as "opus 73 of Beethoven" by *object terms*:

$$op73[composer = beethoven],$$

where $op73$ is a basic object, $composer$ is a label, and $beethoven$ is an object term as the value. There are two kinds of labels: $l \in L_i$ takes a single value and $l^* \in L_s$ takes a set value, where $L_i \cap L_s = \emptyset$. l is called a single value label and l^* is a set value label. Similarly, there are two kinds of variables: $X \in V_i$ (a single value variable) and $X^* \in V_s$ (a set value variable).

An *object term* is defined as follows:

[Def] 1 *Object Term*
Let $o \in B$, $l_1, \cdots, l_n \in L_i$ where $l_i \neq l_j$ $(i \neq j)$, and $t_1, \cdots, t_n$ be object terms or variables ($\in V_i$), then

$$o[l_1 = t_1, \cdots, l_n = t_n] \quad (0 \leq n)$$

is an object term. *When $n = 0$, we simply write o instead of $o[\,]$.* □

For example, $mozart$ and $male[occupation = pianist]$ are object terms. An object term with variables is called a *parametric* object term.

$\preceq$ among basic objects is extended to *subsumption relation* $\sqsubseteq$ among object terms as follows:

[Def] 2 *Subsumption Relation*
Given two object terms without variables, $o[l_1 = t_1, \cdots, l_n = t_n]$ *and* $o'[l'_1 = t'_1, \cdots, l'_m = t'_m]$,
if $o \preceq o'$ *and* $\forall l'_j, \exists l_i \quad l_i = l'_j \wedge t_i \sqsubseteq t'_j$,
then $o[l_1 = t_1, \cdots, l_n = t_n] \sqsubseteq o'[l'_1 = t'_1, \cdots, l'_m = t'_m]$,
where $1 \leq j \leq m$ *and* $1 \leq i \leq n$. □

Example 1 Subsumption Relations
$$apple[color = green] \sqsubseteq apple,$$
$$male[age = 30, occupation = pianist]$$
$$\sqsubseteq person[occupation = musician],$$
where $male \sqsubseteq person$ and $pianist \sqsubseteq musician$. □

The subsumption relation among sets of object terms is also defined. Given two sets of object terms, $\{o_1, \cdots, o_n\}$ $(= S_1)$ and $\{o'_1, \cdots, o'_m\}$ $(= S_2)$, subsumption relation $\sqsubseteq_H$ among sets is defined in Hoare order as follows:
$$S_1 \sqsubseteq_H S_2 \overset{def}{=} \forall o_i \in S_1, \exists o'_j \in S_2 \; o_i \sqsubseteq o'_j$$
Although the Hoare order is not partial, the representative of an equivalence class can be easily defined as a set where any two elements cannot be ordered, and the set of representatives is partially ordered. So we assume without loss of generality that $\sqsubseteq_H$ is a partial order.

Since lattice construction from a partially ordered set is a well known process, we assume that a set O of object terms (without variables) with $\top$ and $\bot$ is a lattice $(O, \sqsubseteq, \top, \bot)$ without loss of generality. The meet and join operations of o_1 and o_2 are denoted by $o_1 \downarrow o_2$ and $o_1 \uparrow o_2$, respectively. Sets of object terms without variables constitute another lattice. Given two sets, S_1 and S_2, we can define meet and join operations ($\Downarrow$ and $\Uparrow$, respectively) under Hoare order as follows:
$$S_1 \Downarrow S_2 \overset{def}{=} \{e_1 \downarrow e_2 | \, e_1 \in S_1, e_2 \in S_2\}$$
$$S_1 \Uparrow S_2 \overset{def}{=} S_1 \cup S_2$$
where $\{\top\}$ is the top of the lattice and $\{\ \}$ is the bottom.

2.2 Subsumption Constraint and Attribute Term

In the previous section, we showed how to represent so-called "is-a" and "a-kind-of" relations between concepts in *QUIXOTE*. To represent relations such as "the first name of Mozart is Amadeus" we use a relation between an object term and a property of another object term:
$$mozart.first\_name \cong amadeus.$$
Given an object term, o, the value of a single value label l is denoted by $o.l$ and the value of a set value label l^* is $o.l^*$. $o.l$ and $o.l^*$ are called *dotted terms*. The term *mozart.first_name* can be read "Mozart's first name." The subsumption constraints of an object term are defined by using dotted terms as follows:

[Def] 3 *Subsumption Constraint*
Let t_1, t_2 *be object terms, single value variables, or dotted terms with single value labels, then* $t_1 \sqsubseteq t_2$ *is a* subsumption constraint. *In the case of a set, if* t^*_1 *and* t^*_2 *are sets of object terms, set value variables, or dotted terms with set value labels, then* $t^*_1 \sqsubseteq_H t^*_2$ *is also a* subsumption constraint. □

When $t_1 \sqsubseteq t_2 \wedge t_1 \sqsupseteq t_2$ $(t^*_1 \sqsubseteq_H t^*_2 \wedge t^*_1 \sqsupseteq_H t^*_2)$, we denote $t_1 \cong t_2$ $(t^*_1 \cong_H t^*_2)$ [1].

A set of subsumption constraints is saturated and reduced by applying the following rewriting rules:
$$\begin{array}{rcl} x \sqsupseteq y & \Rightarrow & y \sqsubseteq x \\ x \sqsubseteq y,\; y \sqsubseteq z & \Rightarrow & x \sqsubseteq y,\; y \sqsubseteq z,\; x \sqsubseteq z \\ x \sqsubseteq y,\; x \sqsubseteq z & \Rightarrow & x \sqsubseteq (y \downarrow z) \\ y \sqsubseteq x,\; z \sqsubseteq x & \Rightarrow & (y \uparrow z) \sqsubseteq x \\ x \cong y,\; y \cong z & \Rightarrow & x \cong y,\; y \cong z, x \cong z \\ x \sqsubseteq y,\; y \sqsubseteq x & \Rightarrow & x \cong y \\ \multicolumn{3}{l}{o[\cdots, l = x, \cdots] \sqsubseteq o'[\cdots, l = y, \cdots]} \\ & \Rightarrow & o \sqsubseteq o', x \sqsubseteq y \end{array}$$
where $x \sqsubseteq x$ and $x \cong x$ are removed in the procedure. The termination and confluency of the above rules are proved in [11]. Similar rules are also defined for set constraints.

[Def] 4 *Attribute Term*
Let $o \in B$, $l_1, \cdots, l_n \in L_i \cup L_s$ *where* $l_i \neq l_j$ $(i \neq j)$, *and* $t_1, \cdots, t_n$ *be object terms or variables* , $op_i \in \{\rightarrow, \leftarrow, =\}$ *(for single value)* $\cup \{\rightarrow_H, \leftarrow_H, =_H\}$ *(for set value) then*
$$o/[l_1 \; op_1 \; t_1, \cdots, l_n \; op_n \; t_n] \quad (0 \leq n)$$
is an attribute term . □

By the following rules, an attribute term is transformed into an object term with subsumption constraints $o|C$, where o is an object term, and C is a set of subsumption constraints.
$$\begin{array}{rcl} o/[l \rightarrow t]\|C & \Longleftrightarrow & o|\{o.l \sqsubseteq t\} \cup C \\ o/[l \leftarrow t]\|C & \Longleftrightarrow & o|\{o.l \sqsupseteq t\} \cup C \\ o/[l = t]\|C & \Longleftrightarrow & o|\{o.l \cong t\} \cup C \\ o/[l^* \rightarrow_H s]\|C & \Longleftrightarrow & o|\{o.l^* \sqsubseteq_H s\} \cup C \\ o/[l^* \leftarrow_H s]\|C & \Longleftrightarrow & o|\{o.l^* \sqsupseteq_H s\} \cup C \\ o/[l^* =_H s]\|C & \Longleftrightarrow & o|\{o.l^* \cong_H s\} \cup C \end{array}$$
An *intrinsic property* is a pair of a label and a value in o. An *extrinsic property* is a subsumption constraint of o in C of an attribute term. If both an intrinsic property and an extrinsic property have the same label, then the extrinsic property is removed. That is,
if $o[\cdots, l = t_1, \cdots]|\{o.l \; op \; t_2\} \cup C$,
then $o.l \; op \; t_2$ is removed,

[1]The semantics of *QUIXOTE* is outlined in three parts:

(1) An object term is mapped into a *labeled graph* as a subclass of a hyperset[2].
(2) The subsumption relation among object terms corresponds to a *bisimulation relation* among labeled graphs.
(3) A label or an object term used as a label corresponds to a *function* on a set of labeled graphs. Here the subsumption relation among labels is not considered.

For details, see [21]

where *op* is $\sqsubseteq, \sqsupseteq$, *or* $\cong$. When a label l does not appear, neither in an intrinsic property nor in an extrinsic property in an attribute term, $o|C$, $\bot \sqsubseteq o.l \sqsubseteq \top$ is assumed.

For *property inheritance*, only extrinsic properties are inherited according to the subsumption relation among object terms as follows:

[Def] 5 *Property Inheritance*
If $o \sqsubseteq p$ and o does not have an intrinsic property of a label l (l^*), then $o.l \sqsubseteq p.l$ and $o.l^* \sqsubseteq p.l^*$. □

That is, by applying a label, which is not included in an intrinsic property of the object terms, the subsumption relation between object terms makes its property inheritance monotonic. Property inheritance *exception* corresponds to the above restriction of extrinsic properties.

Example 2 Property Inheritance

(1) If $apple/[color \to red]$,
then $apple[weight = heavy]/[color \to red]$, but
$apple[color = green]$ does not inherit $color \to red$.

(2) If $apple[weight = heavy]/[area^* \leftarrow_H \{aomori\}]$
and $apple[color = green]/[area^* \leftarrow_H \{nagano\}]$,
then $apple/[area^* \leftarrow_H \{aomori, nagano\}]$
(by the join operation between sets). □

Note that, in the cases of $\leftarrow$ and $\leftarrow_H$, extrinsic properties are inherited upward by the above rule, while intrinsic properties are not, even though $apple[color = green]$ is $apple[color = green]/[color = green]$.

As property inheritance is constraint inheritance in *QUIXOTE*, *multiple inheritance* corresponds to the merging of constraints without preferences.

2.3 Rule and Module

QUIXOTE objects are defined by rules which are modularized into several *modules*.

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

where m is a *module identifier* (mid) (in the form of an object term) and $r_1, \cdots, r_n$ are rules defined below.

For simplicity, we use the notation 'a module m' instead of 'a module with a mid m.' Modules can be nested. When a mid has variables, it is called a *parametric module.* Variables in a mid are global in the module, that is, variables in a mid can be shared by rules in the module. A module can be explicitly referred to by rules in other modules.

2.3.1 Rule

[Def] 6 *Rule*
Let a_0 ($=o_0|C_0$), a_1 ($=o_1|C_1$), $\cdots$, a_n ($=o_n|C_n$) be attribute terms, $m_0, m_1, \cdots, m_n$ be mids, and D a set of subsumption constraints. A rule *is defined as follows:*

$$m_0 :: a_0 \Leftarrow m_1 : a_1, \cdots, m_n : a_n \mid\mid D; ;$$

C_0 must not contain any subsumption relation between object terms. a_0 is called the head *and $m_1 : a_1, \cdots, m_n : a_n \mid\mid D$ is called the* body. □

The rule in the above definition intuitively means that a module m_0 has a rule such that if a_1 is satisfied in a module $m_1, \cdots$, and a_n is satisfied in a module m_n under constraint D, then a_0 is satisfied in a module m_0.

The rule may be transformed:

$$m_0 :: o_0|C_0 \Leftarrow m_1 : o_1, \cdots, m_n : o_n \mid\mid A \cup C; ;$$

where C_0 is called a *head constraint* and $A \cup C = C_1 \cup \cdots \cup C_n \cup D$ is called a *body constraint.* Further, a body constraint can be divided into a set A of constraints containing dotted terms and a set C of the rest of the constraints. The restriction of C_0 in the above definition is to avoid destruction of the lattice by assertion of a subsumption relation during derivation. If a body is empty, the rule is called a *fact.*

From an object-oriented point of view, the rule gives an *intentional* definition of *QUIXOTE* objects. An object in *QUIXOTE* consists of an object term and a set of methods. An object term without variables plays the role of an object identifier (oid)[1, 10], while each extrinsic property plays the role of a method. That is, a label corresponds to a message and the value corresponds to the returned value.

For the case in which there is no head constraint, *QUIXOTE* may be considered as an instance of CLP(X) [7], where a constraint domain is a set of labeled graphs — as a subclass of hypersets[2] and (extended) subsumption relations. Without set subsumption constraints, *QUIXOTE* becomes a subclass of CLP(AFA), with a hyperset constraint domain [11].

2.3.2 Submodule relation and rule inheritance

Module mechanism is introduced with the following objectives:

- modularization and classification of knowledge,
- co-existence or localization of inconsistent knowledge,
- temporal storage of tentative knowledge, and
- introduction of a modular programming style.

To meet these objectives, we define *submodule relation* among modules:

[Def] 7 *Submodule Relations*
Given two modules, m_1 and m_2, a submodule relation $m_1 \sqsupseteq_S m_2$ means that m_1 inherits all the rules in m_2, when m_2 is called a submodule *of m_1.* □

The submodule relation specifies *rule inheritance*, while the subsumption relation specifies property inheritance.

To realize the exception of rule inheritance, each rule can have an inheritance mode *o*, *l*, or *ol*. A rule with *o overrides* inherited rules from the supermodule which have the same head as shown in Example 3. A rule with

l is a *local* rule, which is not inherited by the submodules. Inheritance mode ol is the combination of o and l.

Example 3 The following program describes the knowledge "In Europe, cars usually drive on the right. But cars drive on the left in England."

$england \sqsupseteq_S europe;; france \sqsupseteq_S europe;;$
$europe :: car/[drive = right];;$
$england :: (o)car/[drive = left];;$ □

In the current framework of *QUIXOTE*, the names of object terms, the subsumption relation, and the submodule relation are global in a database, while the existence of objects and extrinsic properties are local. That is, if there is no submodule relation between two modules, then their extrinsic properties do not mutually interfere, that is, inconsistent knowledge can co-exist separately in such modules.

For example, the following program becomes inconsistent because $john$ has a different extrinsic age property in the module $year\_1994$.

$year\_1994 :: john/[age = 20];;$
$year\_1994 :: john/[age = 30];;$

However, the following is not inconsistent, when the submodule relation does not hold between $year\_1982$ and $year\_1994$.

$year\_1982 :: john/[age = 20];;$
$year\_1994 :: john/[age = 30];;$

2.4 Database and Query processing

We define a *database* or a *program* as a triple (S, M, R), where S, M, R correspond to definitions of subsumption relations [2], submodule relations, and rules. Definitions of rules can be considered definitions of objects or definitions of modules.

In reality, a database tends to be partial, that is, necessary definitions might be missing, rules might be ambiguous or indefinite, and a *QUIXOTE* object might be incompletely defined[23]. To treat the partiality of information, *QUIXOTE* has features like hypothetical reasoning and answer with assumption.

In *QUIXOTE*, a query is defined as follows.

[Def] 8 *Query and Answer*
Let $m_0, \cdots, m_n$ *be mids,* $a_0, \cdots, a_n$ *attribute terms, and* C *a set of subsumption constraints,* H *a set of additional rules called* hypotheses. *A query is defined as follows:*

$$? - m_0 : a_0, \cdots, m_n : a_n \parallel C\ [\ ;; H\ ].$$

An answer in QUIXOTE is in the form of

if Assumption *then* Result *because* Explanation

where Assumptions *corresponds to information not in the database,* Result *is a set of subsumption constraints, and* Explanation *shows what knowledge is used to derive the answer.* □

[2]Only the $\preceq$-relation is defined in S.

2.4.1 Hypothetical Reasoning

In general, hypothetical reasoning in a database DB is defined as reasoning in a database $DB \cup H$, where H is a hypothesis [4].

Example 4 Consider the following DB.

$music :: bwv1009/[composer = bach];;$
$listen\_baroque[music = X] \Leftarrow$
$\quad baroque : C, music : X/[composer = C];;$

(listen to a piece composed by a baroque composer.)

For a query $?\text{-}listen\_baroque[music = X]$, the answer is simply no because there are no objects in *baroque* module. For the same query, however, with a hypothesis such that

$? - listen\_baroque[music{=}X];;$
$\quad baroque :: bach/[first\_name = "johan"].$

the answer is $X = bwv1009$. □

To a *QUIXOTE* database (S, M, R), a hypothesis consists of a triple (H_S, H_M, H_R), where H_S, H_M, and H_R are a set of hypotheses for S, M, and R. A query Q with a hypothesis (H_S, H_M, H_R) to a database (S, M, R) is equivalent to a query Q without hypotheses to a database $(S \cup H_S, M \cup H_M, R \cup H_R)$.

In the sequence of queries, such hypotheses are incrementally inserted into a database. To control such insertions, *nested transactions* are introduced into *QUIXOTE*: that is, even if a database is reorganized by hypotheses, the original image is recovered by *rollback* operations.

Example 5 Query Sequence
The following is an example of a query sequence with nested transaction:

?-open_db(DB).	% Open a database named DB
?-begin_trans.	% Begin a transaction (level 1).
?-q_1;;H_1.	% Same as ?-q_1 to $DB \cup H_1$.
?-begin_trans.	% Begin a transaction (level 2).
?-q_2;;H_2.	% Same as ?-q_2 to $DB \cup H_1 \cup H_2$.
?-abort_trans.	% Abort a transaction (level 2).
	% H_2 is rolled back.
?-q_3;;H_3.	% Same as ?-q_3 to $DB \cup H_1 \cup H_3$.
?-end_trans.	% Commit a transaction (level 1).
	% DB is updated to $DB \cup H_1 \cup H_3$.
?-close_db(DB).	% Close a database DB.

□

2.4.2 Answer with Assumption

Example 6 Answer with Assumption

$major \sqsupseteq c\_major;;$
$music :: k551/[type = symphony, name = jupiter];;$
$music :: k467/[type = piano\_concert, key = c\_major];;$
$m :: listen[mood = gloom, music = X] \Leftarrow$
$\quad music : X/[key \rightarrow major];;$

(When gloomy, a piece with a major key is preferable.)

For a query ?-*listen*[$mood = gloom, music = k467$], the answer is simply **yes**. Then, what answer is expected to a query ?-m : *listen*[$mood = gloom, music = k551$] ? Although there is an object $k551$ in *music* module, it does not specify *key* (extrinsic) property. Without making any assumptions, the query fails. However, as we focus on the partiality of the information, the lack of information suggests an assumption be taken. So, in *QUIXOTE*, the answer is that if *music* : $k551.key \sqsubseteq$ *major* then *yes*, that is, unsatisfied constraints of other objects' extrinsic properties in bodies are assumed. □

In logic programming, finding a lack of information or unsatisfiable subgoals corresponds to *abduction*, that is, hypothesis or explanation generation [5]. Remember that a rule in *QUIXOTE* can be represented as follows (see 2.3.1):

$$m_0 :: o_0|C_0 \Leftarrow m_1 : o_1, \cdots, m_n : o_n \mid\mid A \cup C;;$$

In *QUIXOTE*, only dotted term constraints can become assumptions, that is, when body constraints about dotted terms are not satisfied, they are taken as a conditional part of an answer. Although A and C are disjoint, when variables in C are bound by dotted terms during query processing, constraints with the variables in C are moved into A. If the subsumption relation between object terms is taken as assumption, it might destroy the soundness of the derivation because it affects property inheritance and does not guarantee results in the former derivation.

Abduction is closely related to procedural semantics. Here we will only briefly explain the relation [14]. In general, derivation by query processing in CLP is the finite sequence of a pair (G, C) of a set G of goals and a set C of constraints:

$$(G_0, C_0) \Rightarrow (G_1, C_1) \Rightarrow \cdots \Rightarrow (G_{n-1}, C_{n-1}) \Rightarrow (\emptyset, C_n).$$

On the other hand, derivation in *QUIXOTE* is a finite directed acyclic graph of the triple (G, A, C) of the set G of goals, the set A of assumptions, and the set C of constraints.

A query is also transformed in the form of ?-$o_1, \cdots, o_n \mid\mid A_0 \cup C_0$, i.e., a triple $(\{o_1, \cdots, o_n\}, A_0, C_0)$. For a node $(\{G\} \cup G_i, A_i, C_i)$, a rule $G'|C' \Leftarrow B \mid\mid A \cup C$, and $\exists\theta\ G\theta = G'\theta$, where B is a set of object terms and θ is a substitution, the transformed node is:

$$((G_i \cup B)\theta,\ (A_i\theta \setminus C'\theta) \cup A\theta,\ (C_i \cup C \cup C')\theta).^3$$

The derivation image is illustrated in Figure 1. If there are two nodes, (G, A, C) and (G, A', C), where $A \subseteq A'$, then the derivation path of (G, A', C) is thrown away. i.e., only the minimal assumption is made. If there are two nodes, (G, A, C) and (G, A, C'), then they are merged into $(G, A, C \cup C')$.

[3]Note that, here, we ignore that some elements in $(C_i \cup C \cup C')\theta$ might be moved into $(A_i\theta \setminus C'\theta) \cup A\theta$.

2.5 Other New Features of *QUIXOTE*

To describe an actual application, the above basic *QUIXOTE* features are sometimes insufficient. Negative information and arithmetic computation are essential in various experiments. *NAF* (Negation as Failure) and *disequation* constraints are new features to treat negative information. For arithmetic calculation and comparison, *QUIXOTE* is equipped with a special built-in module called *math_module* mechanism.

2.5.1 Negation as Failure

One of the crucial defects of *QUIXOTE* is that it cannot represent negative information such as "if there does not exist some object, then ...." or "if a certain relation does not hold, then ...."

One of the new feature of *QUIXOTE* is NAF (Negation As Failure). As with NAF in Prolog, when the derivation of a goal (object) fails, negation of the goal is considered to have succeeded. Using NAF, the user can check for the existence of an object in a module. For the following example describing "the ticket price is 1000 yen for members and 1500 yen for non-members."

$$m :: ticket/[price = 1000] \Leftarrow is\_member;;$$
$$m :: ticket/[price = 1500] \Leftarrow \neg is\_member;;$$

when there exists an object *is_member* in module m, $? - m : ticket/[price = X]$. has an answer $X = 1000$. If *is_member* does not exists, the answer is $X = 1500$. [4]

2.5.2 Element_of and Disequation constraints

For actual applications, there are constraints like "a variable can be bound to mozart, beethoven, or bach." or "a variable does not have the same value of another variable." To treat such knowledge, *QUIXOTE* has element_of and disequation constraints.

[Def] 9 *Element_of Constraint*
Let t be object terms, single value variables, or dotted terms with single value labels, s be a set of ground object terms, then $t \in s$ is an element_of constraint. □

[Def] 10 *Disequation Constraint*
Let t_1, t_2 be ground object terms, single value variables, or dotted terms with single value labels, then $t_1 \neq t_2$ is a disequation constraint. □

[4]In an implementation (big-*QUIXOTE*, see 3.1), the *QUIXOTE* program including NAF must be stratified; that is, every object in a program can be attached to a level value (non-negative integer) so as to meet the following condition:

> In every rule $h \Leftarrow B$, the positive objects in B have equal or lower levels than h and negative goals have lower levels than h.

$$\begin{tikzcd}[column sep=small]
\vdots & \cdots \arrow[r] & (G_i,A_i,C_i) \arrow[dr] & & & \\
(G_0,A_0,C_0) \arrow[ur] \arrow[dr] & & & (G_k,A_k,C_k) \arrow[r] & \cdots \arrow[r] & (\emptyset,A_n,C_n) \\
\vdots & \cdots \arrow[r] & (G_j,A_j,C_j) \arrow[ur] & & &
\end{tikzcd}$$

Figure 1: Derivation Network

In addition to rewriting rules in 2.2, the following rules are added to treat element_of and disequation constraints.

$$\begin{aligned} x \in s1,\ x \in s2 &\Rightarrow x \in s1 \cap s2 \\ x \in \{a\} &\Rightarrow x \cong a \\ x \neq a,\ x \in s &\Rightarrow x \in s - \{a\} \end{aligned}$$

Constraints such as $x \neq y$ (x and y are not unifiable) and $o \in \{o, \cdots\}$ are removed. When $x \neq x$, $o \in s$ (o is ground and s does not contain o), or $x \in \{\}$ occurs in the constraint rewriting process, the constraints are unsatisfiable.

2.5.3 Math module

Knowledge in actual applications contains various domains of constraints other than subsumption ones, such as arithmetic, combinatorial, and so on. To cope with such variety of constraints, heterogeneous cooperative problem solving system Helios is under development in ICOT[3]. In Helios, various constraint solvers, DBs, and applications are wrapped by a capsule mechanism to become communicating agents.

A special built-in module called *math module* is designed as an experimental device to call external constraints in *Quixote*, which allow calculating and comparing of simple numerical expressions (Example 7). *math* module offers several built-in predicates as follows to calculate and compare numerical expressions.

$add(A1, A2, R)$:	$R \leftarrow A1 + A2$
$subtract(A1, A2, R)$:	$R \leftarrow A1 - A2$
$multiply(A1, A2, R)$:	$R \leftarrow A1 * A2$
$less\_than(A1, A2)$:	check $A1 < A2$
etc.	

The evaluation of a goal in *math* module is delayed until proper arguments ($A1$ and $A2$ in the above predicates) are bound to ground terms.

Example 7 Math module
The following program calculates at what age a musician composes a piece of music.

$$\begin{aligned}
&composer :: mozart/[born = 1756, dead = 1791];; \\
&music :: k467/[year = 1785, composer = mozart];; \\
&music :: X/[age\_of\_composer = A] \Leftarrow \\
&\qquad music : X/[year = Y, composer = C], \\
&\qquad composer : C/[born = B], \\
&\qquad math : subtract(Y, B, A);;
\end{aligned}$$

□

3 Implementation

This section explains implementation issues of *Quixote*. Until now, there have been two kinds of implementation: big-*Quixote* and micro-*Quixote*. Both system are registered as ICOT Free Software (IFS) [5].

3.1 big-*Quixote*

big-*Quixote* is an implementation of *Quixote* language described in Section 2, which consists of *Quixote*-client and *Quixote*-server modules as shown in Figure 2.

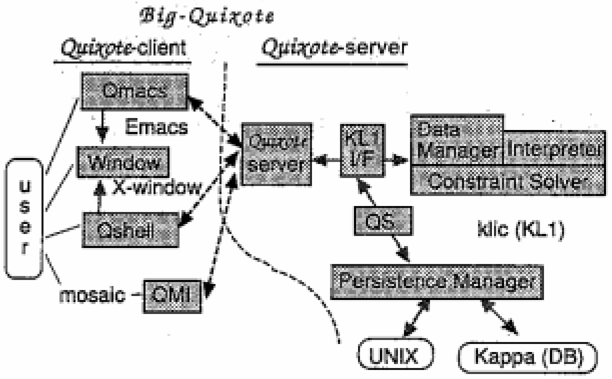

Figure 2: System configuration of big-*Quixote*

Quixote-server, which is composed of the following modules, is mainly implemented in the KL1 language on PIMOS and KLIC systems.

1. *Quixote* server: TCP/IP communication interface.
2. KL1 IF: data transformation, etc.
3. QS: manages external DBs.
4. Persistence Manager: interface to external DBs.
5. Data Manager: manages internal representation of *Quixote* objects.
6. Interpreter: makes inference.
7. Constraint Solver: solves subsumption, set, and disequation constraints.

In the latest version of big-*Quixote* (ver.4), there are three kinds of client interface: Qmacs, Qshell, and mosaic and an X-Windows interface.

[5]ICOT free software is available by anonymous FTP from `ftp.icot.or.jp`

1. Qmacs: interactive user interface on top of GNU-Emacs.
2. Qshell: batch user interface with QIF libraries.
3. QMI: interface between *QUIXOTE* and mosaic.
4. window: window interface to display lattice structures, module hierarchies, and derivation trees.

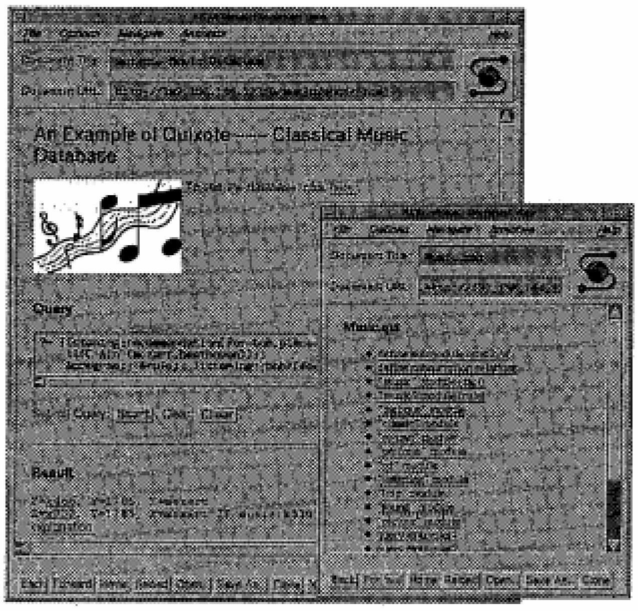

Figure 3: Mosaic interface of big-*QUIXOTE*

Figure 3 shows the mosaic interface of big-*QUIXOTE* through QMI.

3.2 micro-*QUIXOTE*

Because the *QUIXOTE* language has various features, its full implementation tends to be too heavy to run a small program. micro-*QUIXOTE* is designed to extract central features of *QUIXOTE* as a programming language and offers a small system for knowledge information processing[13]. micro-*QUIXOTE* supports the following feature of *QUIXOTE*.

- object terms (without set),
- subsumption constraints,
- property inheritance,
- module, and
- answer with assumptions, hypothetical reasoning.

For simplicity, micro-*QUIXOTE* utilizes a Prolog-like depth-first search without merging derivations. Accordingly, micro-*QUIXOTE* sometimes returns different answers from big-*QUIXOTE*.

micro-*QUIXOTE* is implemented in the C language independent of big-*QUIXOTE* and has the following features.

- Everything is implemented in C and has high portability,
- small system (199KB of source code), and
- external call mechanism.

System configuration of micro-*QUIXOTE* is shown in Figure 4.

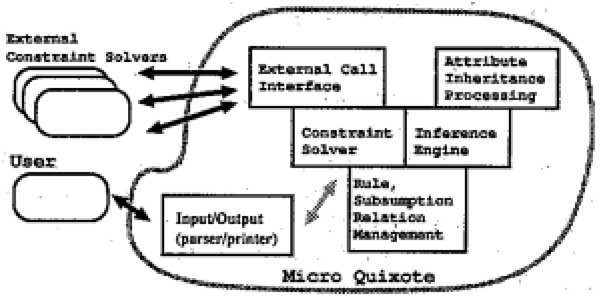
External Constraint Solvers
User
External Call Interface
Attribute Inheritance Processing
Constraint Solver
Inference Engine
Rule, Subsumption Relation Management
Input/Output (parser/printer)
Micro Quixote

Figure 4: System configuration of micro-*QUIXOTE*

micro-*QUIXOTE* allows the representation of external constraints, whose operators begin and end with "#", such as

```
?-X/[name=A] ||
      {X=<male, A #regexp# ''on''}.
```
(search a man whose name contains "on.")

When an external constraint binds to ground, micro-*QUIXOTE* throws it out and waits for the result (true or false). This is the external call mechanism. In Figure 4, external call messages are dispatched in GNU-Emacs to the windows interface or bc (binary calculator), etc. External call mechanism plays an important role when micro-*QUIXOTE* is embedded into the heterogeneous cooperative problem solving system Helios [3].

4 Application

We have developed several applications in *QUIXOTE*. Among these, this section introduces three kinds of application: legal reasoning, genetic information processing, and natural language processing. Details are reported in other papers [19, 16, 17, 22].

4.1 Legal Reasoning

Our legal reasoning system [19, 22] 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*. Usually, the analytical legal reasoning process consists of three steps: *fact finding*, *statutory interpretation*, and *statutory application*. Among them, we focus on statutory applications, which can be considered:

analogy detection: Given a new case, similar precedents to the case are retrieved from existing precedents.

rule transformation: Precedents (interpretation rules) extracted by analogy detection are abstracted until the new case can be applied to them.

deductive reasoning: Apply the new case in a deductive manner to abstracted interpretation rules transformed by rule transformation.

In these three steps, the analogy detection strategy is essential in legal reasoning for *more efficient* detection of *better* precedents, which decides the quality of the results. To investigate the *QUIXOTE*'s potential for legal reasoning, we developed an experimental system [19, 22]. Here we describe a simplified example of legal reasoning.

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.

4.2 Biological 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 the information partiality mentioned in 2.4, all the complete and consistent 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 such as Kappa [25], 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*.

The following three examples show how *QUIXOTE* is used in biological information processing.

1. *Description of protein functions and motifs:*
   Chemical reactions correspond to rules and the chains correspond to transitive rules.

2. *Description of experimental data:*
   For example [16], 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. *Inconsistent experimental data:*
   A module in *QUIXOTE* is appropriate for storing and classifying data with hypotheses.

4.3 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.

We first define a *situated inference* rule as follows:

$$s_0 \models \sigma_0 \Leftarrow s_1 \models \sigma_1, s_2 \models \sigma_2, \cdots, s_n \models \sigma_n \| C. \quad (1)$$

This sample rule can be interpreted as: if s_1 supports σ_1, s_2 supports σ_2, and so on, thus we can infer that s_0

supports σ_0, under constraint C. Concepts in situation theory are rephrased in *QUIXOTE* as follows [17, 22]:

situation theory		*QUIXOTE*
situation	$\Leftrightarrow$	module
infon	$\Leftrightarrow$	object term
role	$\Leftrightarrow$	label
supporting ($\models$)	$\Leftrightarrow$	membership (:)

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 Example 8. Details in the query processing are reported in [18].

Example 8 Quine's example

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

□

For a query

$?- speaker\_a : bizet/[nationality = X]$,

the answer is that

if $bizet.nationality = verdi.nationality$
and $verdi.nationality = italy$
then $X = italy$.

On the other hand, for a query

$?- speaker\_b : verdi/[nationality = X]$,

the answer is that

if $verdi.nationality = bizet.nationality$
and $bizet.nationality = france$
then $X = france$.

Even if there is insufficient data in a database, *QUIXOTE* explicates hidden knowledge.

The query processing process is more complex than conventional processing because modes may be merged. Additionally, we might get different answers from the same query if the query is made to different modules. In such cases, the derivation process of an answer is returned including an *explanation*, if necessary. By referring to this explanation, users can verify which rules are used for the answer.

5 Concluding Remarks

This paper describes overviews of *QUIXOTE* from language to implementation issues and applications. Here, we compare *QUIXOTE* with related works. It resembles F-logic [8] in the sense that it is a DOOD language and introduces object-orientation concepts into logic programming. Though, as a knowledge representation language, *QUIXOTE* has additional convenient features such as a module mechanism, abductive inference, and so on. Unlike the conventional CLP language [7], *QUIXOTE* has a symbolic constraint domain and abductive reasoning mechanism which are suitable for various knowledge processing application. The concept of a complex object in *QUIXOTE* inherits PST (partially specified term) in CIL [12], that is another ancestor language developed in ICOT.

The features of *QUIXOTE* is summarized as follows:

- object-orientation concepts such as object identity and property inheritance are introduced into the logic programming as the fundamental philosophy,
- the concept of module enables us to have local definition in a large knowledge-base,
- its logical inference system is extended to be based on abduction and hypothetical reasoning,
- employs several peripheral mechanisms for application: such as NAF, disequation constraints, math module, and external call mechanism, and
- two kinds of implementation: big-*QUIXOTE* and micro-*QUIXOTE*.

To show the effectiveness, we applied *QUIXOTE* to legal reasoning, natural language processing, and biological database, and so on. Especially *QUIXOTE* features to treat partial information, such as property, constraint, and hypothesis/assumption-based Q&A, play important roles in the above applications.

There still remains, however, some future research topics, if we make *QUIXOTE* a more efficient and applicable knowledge representation language. For example, in legal reasoning, there often occurs information such as modal representation and external (especially temporal) constraints [15]. Modal representation is information using modal operator (*must*, *may*, *can*, and so on), such as "there may exists an object" and "an attribute of an object must be something." If such modality is embedded in the module and constraint, $\mathcal{QUIXOTE}$ will be useful in not only describing legal knowledge but treating semantics in natural language processing in general.

Temporal information also occurs in various applications; such as "event A occurred earlier than event B" and "event C occurred during event D." Besides temporal constraints, knowledge in actual applications is usually composed of various heterogeneous information as suggested in 2.5.3. To cope with such variety, it is not a promising approach to extend $\mathcal{QUIXOTE}$ itself every time to handle the appropriate domain of constraints. Another way to tackle the heterogeneity is to combine $\mathcal{QUIXOTE}$ with other constraint solvers and databases and solve a problem cooperatively, as introduced in Helios[3, 25]. As there already exist various constraint solvers, databases, and knowledge representation languages, such a distributed and cooperative approach is becoming important. The math module in $\mathcal{QUIXOTE}$ and the external call mechanism in micro-$\mathcal{QUIXOTE}$ give an experimental view of knowledge processing in distributed and integrated environments.

As for a DB query system, an important feature of the $\mathcal{QUIXOTE}$ system is that the user can ask to a database with hypotheses and get the answer with certain assumptions. By repeating the exchange under nested transaction mechanism, the user can freely add and extract knowledge to and from the $\mathcal{QUIXOTE}$ system. $\mathcal{QUIXOTE}$ provides an experiment device of knowledge handling. One of important topics missing in the current $\mathcal{QUIXOTE}$ implementation is the programming environment. When constructing a $\mathcal{QUIXOTE}$ database, the user has to consider various aspects concerning objects; which concepts in the application correspond to objects, what is the subsumption relation, and how objects are stored locally in several modules, and so on. Programming environment that supports such design methodology is required.

Acknowledgments
The authors thank to all the members who are/were engaged in the $\mathcal{QUIXOTE}$ project now and in the past, for their valuable comments, encouragements, and efforts to develop the $\mathcal{QUIXOTE}$ system.

A Example: music.qxt

This is a sample program, a classical music database, in big-$\mathcal{QUIXOTE}$. $\sqsupseteq$, $\sqsupseteq_S$, $\rightarrow$, and $\cong_H$ are described as >=, >-, ->, and =*= respectively. A rule with the delimiter ';' is called a serialized rule and its body is processed from left to right sequentially. In the current version of big-$\mathcal{QUIXOTE}$, the user has to serialize the goal order when using the math module.

```
&program;;
  %%%%  definition of submodule relations
  &submodule;;
  composer >- baroque+classic+roman;;
baroque>-picture;;classic>-picture;;roman>-picture;;
  music >- sound;;
  %%%%  definition of subsumption relations
  &subsumption;;
  concert >= {piano_concert, violin_concert};;
  instrument >= {stringed,wind,percussion};;
  stringed >= {violin, viola,cello};;
  orchestral >= {violin, viola, oboe};;
  major >= {c_major ,d_major};;
  person >= {john, bob, ken};;
  ....
  %%%%  facts and rules.
  &rule;;
  %% music database
  music::k466/[composer=mozart, type=piano_concert,
         no=20, year=1785,key=d_minor];;
  music::k467/[composer=mozart, type=piano_concert,
         no=21, year=1785, key=c_major];;
  music::op108[composer=beethoven]/
           [type=symphony, no=9, name="coral"];;
         ......
  %% composer database
  baroque::bach/[born=1685, dead=1750,
       first_name="Johan",middle_name="Sebastian",
       last_name="Bach"];;
  classic::mozart/[born=1756, dead=1791,
       first_name="Wolfgang",
       middle_name="Amadeus", last_name="Mozart"];;
       .......
  %% performance database
  perform::k466[pianist=gulda,conductor=abbado,
      orchestra=wpo, year=1975]/
      [cd=cd[no=1],track=1];;
  perform::k385[conductor=walter,orchestra=cso,
       year=1959]/[cd=cd[no=2],track=5];;
  ....
  %%%%% rules
  %% "A major key piece sounds cheerful."
  listening::cheerful[piece=X]
        <= music:X/[key->major];;
  listening::gloom[piece=X]
        <= music:X/[key->minor]};;
  %% "I recommend cheerful pieces to gloomy people."
  listening:: recommendation[for=A, piece=X]
        <= A/[feeling->gloom],
           cheerful[piece=X]
           || {A =< person};;
%% Piano concerts are played by piano and orchestra.
  music::X/[instruments* =*= {piano,orchestral}]
        <= X/[type=piano_concert];;
  %% year <-> age (serialized rule)
  music::X/[age_of_composer=A] <=
         X/[year=Y, composer=C]; C/[born=B];
         math:subtract(Y,B,A);;
  %%%%% sound and picture resource files
  sound::k551/[soundtype=au,
     soundfile="/home/qxt/sound/jupiter.au"];;
picture::bach/[picttype=jpeg,
     pictfile="/home/qxt/picture/bach.jpeg"];;
     ......
&end.
```

References

[1] S. Abiteboul and P. Kanellakis, "Object Identity as a Query Language Primitive," *Proc. ACM SIGMOD International Conference on Management of Data*, Portland, June, 1989.

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

[3] A. Aiba, K. Yokota and H. Tsuda, "Heterogeneous Distributed Cooperative Problem Solving System Helios" *Proc. International Conference on Fifth Generation Computer Systems*, ICOT, 1994.

[4] N. Cercone and G. McCalla (eds.), *The Knowledge Frontier — Essays in the Representation of Knowledge*, Springer-Verlag, 1987.

[5] E. Charniak and D. McDermott, *Introduction to Artificial Intelligence*, Addison-Wesley, 1985.

[6] C. Delobel, M. Kifer, and Y. Masunaga (eds.), *Deductive and Object-Oriented Databases*, (*Proc. the Second International Conference on Deductive and Object-Oriented Databases (DOOD'91)*), LNCS 566, Springer, 1991.

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

[11] 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.

[12] K. Mukai and H. Yasukawa. "Complex Indeterminates in Prolog and its Application to Discourse Models", *New Generation Computing*, 3(4):441–466, 1985.

[13] Y. Niibe, C. Takahashi, and K. Yokota, "Design and Implementation of micro-*QUIXOTE* and Its Extension Function", *Proc. Joint Workshop of SIGDBS of IPSJ and SIGDE of IEICE*, July 139-146, 1994 (in Japanese).

[14] T. Nishioka, R. Ojima, 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 (in Japanese).

[15] C. Takahashi and K. Yokota, "Constructing a Legal Database on *QUIXOTE*", Proc. the Sixth Australasian Database Conference (ADC'95), Adelaide, Australia, Jan. 30,31, 1995.

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

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

[18] S. Tojo, H. Tsuda, H. Yasukawa, K. Yokota, and Y. Morita, "*QUIXOTE* as a Tool for Natural Language Processing," *Proc. the Fifth International Conference on Tools with Artificial Intelligence*, Boston, Nov. 8-11, 1993.

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

[20] H. Yasukawa, H. Tsuda and K. Yokota, "Object, Properties and Modules in *QUIXOTE*," *Proc. International Conference on Fifth Generation Computer Systems*, ICOT, Tokyo, June 1-5, 1992.

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

[22] 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.

[23] K. Yokota, Y. Morita, H. Tsuda, H. Yasukawa, and S. Tojo, "Query Processing for Partial Information Databases in *QUIXOTE*", *ICTAI94*, 1994.

[24] K. Yokota, H. Tsuda, and Y. Morita, "Specific Features of a Deductive Object-Oriented Database Language *QUIXOTE*," *Proc. ACM SIGMOD Workshop on Combining Declarative and Object-Oriented Databases*, Washington DC, USA, May 29, 1993.

[25] K. Yokota, "From Databases to Knowledge-Bases — Kappa, *QUIXOTE*, Helios", *Proc. Int. Symp. on FGCS (FGCS'94)*, Dec, 1994.