PROCEEDINGS OF THE INTERNATIONAL CONFERENCE
ON FIFTH GENERATION COMPUTER SYSTEMS 1992,
edited by ICOT.

Towards an Integrated Knowledge-Base Management System

Overview of R&D on Databases and Knowledge-Bases in the FGCS Project

Kazumasa Yokota Hideki Yasukawa
Third Research Laboratory
Institute for New Generation Computer Technology (ICOT)
4-28, Mita 1-chome, Minato-ku, Tokyo 108, Japan
Tel: +81-3-3456-3069 Fax: +81-3-3456-1618
{kyokota,yasukawa}@icot.or.jp

Abstract

Knowledge representation languages and knowledge-bases play a key role in knowledge information processing systems. In order to support such systems, we have developed a knowledge representation language, *Quixote*, a database management system, *Kappa*, as the database engine, some applications on *Quixote* and *Kappa*, and two experimental systems for more flexible control mechanisms.

The whole system can be considered as under the framework of deductive object-oriented databases (DOODs) from a database point of view. On the other hand, from the viewpoint of the many similarities between database and natural language processing, it can also be considered to support situated inference in the sense of situation theory. Our applications have both of these features: molecular biological databases and a legal reasoning system, TRIAL, for DOOD and a temporal inference system for situated inference.

For efficient and flexible control mechanisms, we have developed two systems: cu-Prolog based on unfold/fold transformation of constraints and dynamical programming based on the dynamics of constraint networks.

In this paper, we give an overview of R&D activities for databases and knowledge-bases in the FGCS project, which are aimed towards an integrated knowledge-base management system.

1 Introduction

Since the Fifth Generation Computer System (FGCS) project started in 1982, many knowledge information processing systems have been designed and developed as part of the R&D activities in the framework of logic and parallelism. Such systems have various data and knowledge, that is, expected to be processed efficiently in the form of databases and knowledge-bases such as electronic dictionaries, mathematical databases, molecular biological databases, and legal precedent databases [1]. Representing and managing such large amounts of data and knowledge for these systems has been a major problem. Our activities on databases and knowledge-bases are also devoted to such data and knowledge under logic paradigm.

Since the late seventies, many data models have been proposed for extension of the relational model in order to overcome various disadvantages such as inefficient representation and inadequate query capability. Among their extensions, *deductive databases* attracted many researchers not only in logic communities but also in artificial intelligence communities, because of its logic platform and strong inference capability. Many efforts on deductive databases have defined the theoretical aspects of databases and have showed the powerful capability of query processing. However, from an application point of view, the data modeling capability is rather poor. This is mainly due to representation based on first-order predicates, which inherits the disadvantages of the relational model. On the other hand, *object-oriented databases* have become popular among extensions of the relational model for coping with 'new' applications such as CAX databases and multi-media databases. The flexibility and adaptability of object-orientation concepts should be examined also in the context of deductive databases, even if object-oriented databases have disadvantages such as poor formalism and semantic ambiguity.

[1] The boundary between *databases* and *knowledge-bases* is unclear and their usage depends on context. Most database communities prefer to use the term *database* even if databases store a set of rules and have an inference capability such as deduction and abduction: e.g., deductive databases, expert databases, and self-organizable databases. In this paper, we also use the term *database* according to this convention. The term *knowledge-base* in our title shows our view that an approach based on extensions of databases is a better way to *real* knowledge-bases than based on conventional *knowledge-bases* used by expert systems.

As it is appropriate for advanced applications to integrate their advantages, we proposed *deductive (and) object-oriented databases* (DOODs) [Yokota and Nishio 1989] [2] , where extensions of the relational model (or deductive databases and object-oriented databases) are considered from three directions: logic, data model, and computational model. The DOOD can be said to be a framework for such extensions. On the other hand, considering the many similarities between DOODs and natural language processing, the framework is also appropriate for situated inference in natural language processing. Such an observation leads us firmly towards an integrated knowledge-base management system over databases and knowledge representation languages.

In the FGCS project, we focus on DOODs as the target of knowledge-base management systems, based on the above observation, and have developed a knowledge-base (or knowledge representation) language *Quixote*, its database engine *Kappa*, and their applications. *Quixote* is a DOOD language. Also, a DOOD system based on *Quixote* has been implemented. We outline their features in Section 2. In order to process a large amount of data efficiently in the DOOD system, there should be a database engine at the lower layer. The engine is called *Kappa*, the data model of which is a nested relational model as a subclass of DOODs. For more efficient processing, a parallel database management system, *Kappa-P*, has been implemented on parallel inference machines. The data model and system are described in Section 3. We are also developing some applications on the DOOD system: a legal reasoning system (TRIAL), a molecular biological database, and a temporal inference system in natural language processing. An overview is given in Section 4. Together with the above works, we are engaged in R&D on more flexible control of logic programs: constraint transformation and dynamical programming, which are expected to be embedded in *Quixote*. We explain these in Section 5. Their relationship is shown in Figure 1. Finally we describe related works and future plans for further extensions of our knowledge-base management system.

2 Knowledge Representation Language (*Quixote*)

Our approach to knowledge-bases follows the previously mentioned deductive object-oriented databases (DOOD). The language, called *Quixote*, designed for the objective has various features [3] : a constraint logic programming language, a situated programming language, object-oriented database programming language, and a DOOD language, besides the features appearing in Figure 1.

2.1 Basic Concepts

Consider the example[Yoshida 1991] in Figure 2 for the genetic information processing system. In the figure,

```
object(ref('Patterson et al (1981)').
       1991/4/24.
       [kind(paper).
        authors(['D. Patterson',
                 'S. Graw',
                 'C. Jones'
                 ]).
        title('Demonstration by somatic cell genetics of
             coordinate regulation of genes for two
             enzymes of puring synthesis assigned to
             human chromosome 21').
        journal('Proc. Natl. Acad. Sci. USA').
        volume(78).
        pages(405-409).
        year(1981)
       ]).
```

Figure 2: A Record (Term) in Prolog

the third argument of the term is peculiar: a tuple (in the form of a list) consisting of pairs of a label and its value. The *author* label has a set value, also, in the form of a list and some values might have a more complex value (another tuple). User programs must be responsible for such structure and unification among these terms. The reason why such a structure is necessary is that a record type (a scheme) cannot be decided in advance. That is, we can get only partial information for an object, because the object itself is not stable, generally. Such characteristics do not necessarily allow application of conventional normalization in the relational model to the design. By introducing an identity concept, such a record can be represented in the form of a set of binary relations, each of which has an identifier, however this is too inefficient in representation.

In *Quixote*, we introduce the concepts of an *object identifier* (*oid*) and a *property*, both of which are based on *complex object* constructors. The example in Figure 2 can be represented as in Figure 3 in *Quixote*. In the figure, the left hand side of "/" is an oid (called an *object term* (*o-term*) in *Quixote*) and the right hand side is the related properties. An *object* consists of an oid and its properties, and can be written as a set of *attribute terms* (*a-terms*) in *Quixote* with the same oid as follows:

$$o/[l_1 = a, l_2 = b] \iff o/[l_1 = a],\ o/[l_2 = b]$$

[2] International conferences were held in Kyoto and Munich [Kim *et al.* 1990, Delobel *et al.* 1991] to work towards such integration.

[3] See the details in [Yasukawa *et al.* 1992].

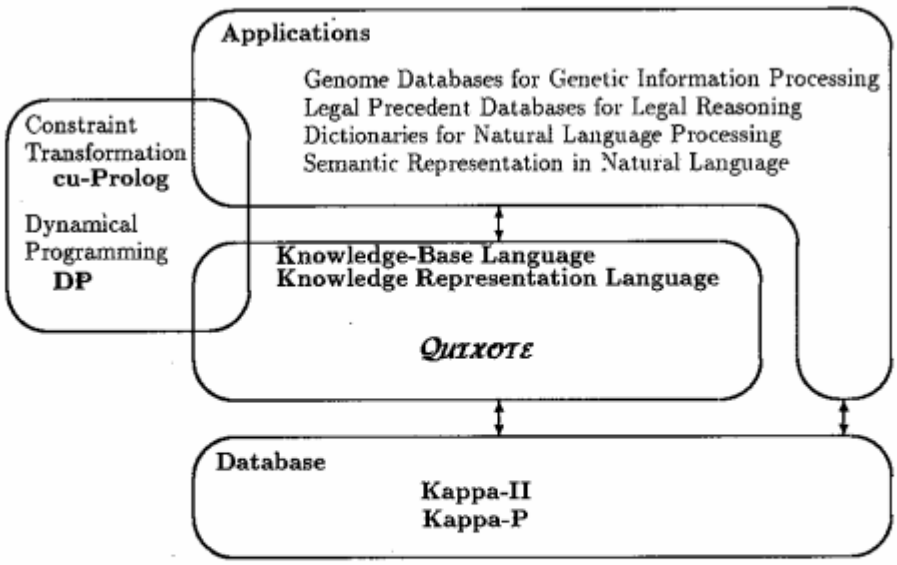


Figure 1: The Framework of a Knowledge-Base (DOOD) Management System of the FGCS Project

```
object[ref='Patterson et al (1981)']/
       [date=1991/4/24,
        kind=paper,
        authors={'D. Patterson',
                 'S. Graw',
                 'C. Jones'
                },
        title='Demonstration by somatic cell genetics of
               coordinate regulation of genes for two
               enzymes of puring synthesis assigned to
               human chromosome 21',
        journal='Proc. Natl. Acad. Sci. USA',
        volume=78,
        pages=405-409,
        year=1981
       ].
```

Figure 3: An Object in QUIXOTE

Such description is effective for processing partial information. Attributes in an o-term are intrinsic for the object:

$$o[l=c]/[l_1=a, l_2=b] \Longleftrightarrow o[l=c]/[l=c, l_1=a, l_2=b]$$

where the right hand side, "$[\cdots]$", of "/" is called the *attribution* of $o[l=c]$. An attribute in an o-term is called an *intrinsic* (or immutable) attribute and an attribute appearing only in the attribution is called an *extrinsic* (or mutable) attribute [4].

[4]We sometimes abuse the terms *attribute* and *property*. Although both an *attribute* and a *property* are, usually, a pair of a label and a (possibly complex) value, an *attribute* is frequently used in the context of record structure, while a *property* is frequently used in the context of object structure. In QUIXOTE, a pair of a label and a value (or a triple of a label, an operator, and a value) is called an *attribute*, however, in the context of inheritance, we use *property* inheritance as a convention. As only extrinsic attributes are inherited in QUIXOTE, as mentioned later, extrinsic attributes are simply called properties. Furthermore, there is a case where an attribute means only a label, as in an attribute-value pair, the meaning, however, is usually clear in the context.

Another problem is the expressive power of oids and properties. First, along the style of logic programming, an oid can be defined intensionally by a set of rules as follows:

$$path[from=X, to=Y] \Leftarrow arc[from=X, to=Y].$$
$$path[from=X, to=Y] \Leftarrow arc[from=X, to=Z], path[from=Z, to=Y].$$

In this program, $path[from=X, to=Y]$ is transitively defined from a set of facts such as $arc[from=a, to=b]$ and so on, and the oid is generated by instantiating X and Y as the result of execution of the program. This guarantees that an object can have a unique oid even if the object is generated in different environments. Furthermore, in order to define a circular path, we must introduce a *tag* and represent a, so-called, complex object with a set and a tuple constructor.

$$X:o[l=X] \Longleftrightarrow X|\{X=o[l=X]\}$$
$$o[l=\{a, \cdots, b\}] \Longleftrightarrow o[l=a] \wedge \cdots \wedge o[l=b]$$

The first example shows that a variable X is an oid with a constraint, $X=o[l=X]$. The second shows that a set in an o-term can be decomposed into conjunction of o-terms without a set constructor.

On the other hand, properties might be indefinite, that is, only in the form of constraints. We introduce the following operators between a label and a (set of) value, and transform them into a set of constraints by introducing dot notation:

$$\begin{aligned} o/[l=a] &\iff o/|\{o.l \cong a\} \\ o/[l\rightarrow a] &\iff o/|\{o.l \sqsubseteq a\} \\ o/[l\leftarrow a] &\iff o/|\{o.l \sqsupseteq a\} \\ o/[l=\{a,\cdots,b\}] &\iff o/|\{o.l \cong_H \{a,\cdots,b\}\} \\ o/[l\rightarrow\{a,\cdots,b\}] &\iff o/|\{o.l \sqsubseteq_H \{a,\cdots,b\}\} \\ o/[l\leftarrow\{a,\cdots,b\}] &\iff o/|\{o.l \sqsupseteq_H \{a,\cdots,b\}\} \end{aligned}$$

The right hand side of "/|" is a set of constraints about properties, where $\sqsubseteq_H$ and $\sqsupseteq_H$ are a partial order generated by Hoare ordering defined by $\sqsubseteq$ and $\sqsupseteq$, respectively, and $\cong_H$ is the equivalence relation. If an attribute of a label l is not specified for an object o, o is considered to have a property of l without any constraint.

The semantics of oids is defined on a set of labeled graphs as a subclass of hypersets[Aczel 1988]: an oid is mapped to a labeled graph and an attribute is related to a function on a set of labeled graphs. In this sense, attributes can be considered *methods* to an object as in F-logic[Kifer and Lausen 1989].

The reason for adopting a hyperset theory as the semantic domain is to handle an infinite data structure. The details can be found in [Yasukawa *et al.* 1992].

2.2 Subsumption Relation and Property Inheritance

Given a partial order relation in a set of basic (non-structural) objects, we can constitute a lattice in a set of ground object terms, the order of which is called the *subsumption relation* $\sqsubseteq$. This is already used as a relation for properties as constraints. According to the relation, properties are inherited downward and/or upward among objects. A general *property inheritance* rule is as follows:

$$o_1 \sqsubseteq o_2 \ \supset\ o_1.l \sqsubseteq o_2.l$$

where intrinsic attributes are out of inheritance. According to the rule, we can get the following:

$$\begin{aligned} o_1 \sqsubseteq o_2,\ o_2/|\{o_2.l \sqsubseteq a\} &\implies o_1/|\{o_1.l \sqsubseteq a\} \\ o_1 \sqsubseteq o_2,\ o_1/|\{o_1.l \sqsupseteq a\} &\implies o_2/|\{o_2.l \sqsupseteq a\} \\ o_1 \sqsubseteq o_2 \sqsubseteq o_3,\ o_2/|\{o_2.l \cong a\} &\implies o_1/|\{o_1.l \sqsubseteq a\}, \\ & \qquad\quad o_3/|\{o_3.l \sqsupseteq a\} \end{aligned}$$

where it can be noted that $o_2/|\{o_2.l \sqsubseteq a\}$ is $o_2/[l\rightarrow a]$: that is, property inheritance is constraint inheritance. In complex o-terms, intrinsic attributes cause the exception of property inheritance:

$$o[l=a] \sqsubseteq o,\ o/[l\rightarrow b] \implies o[l=a]/[l=a]$$

Multiple inheritance is defined upward and downward as the merging of constraints:

$$\begin{aligned} o_1 \sqsubseteq o_2,\ o_1 \sqsubseteq o_3,\ o_2/[l\rightarrow a],\ o_3/[l\rightarrow b] & \\ \implies o/[l \rightarrow \mathrm{meet}(a,b)] & \\ o_1 \sqsupseteq o_2,\ o_1 \sqsupseteq o_3,\ o_2/[l\leftarrow a],\ o_3/[l\leftarrow b] & \\ \implies o/[l\leftarrow \mathrm{join}(a,b)] & \end{aligned}$$

where a set of constraints are reduced by the constraint solver.

2.3 Program and Database

A *module* concept is introduced in order to classify knowledge and handle (local) inconsistencies. Let m be a *module identifier* (*mid*) (syntactically the same as an o-term) and a be an a-term, then $m:a$ is a *proposition*, which means that m *supports* a. Given a mid m, an a-term a, and propositions $p_1,\cdots,p_n$, a *rule* is defined as follows:

$$m :: a \Leftarrow p_1,\cdots,p_n.$$

which means that a module with a mid m has a rule such that if $p_1,\cdots,p_n$ hold, a holds in a module with a mid m. If a mid is omitted in p_i, m is taken as the default and if m is omitted, the rule is held in all modules. a is called a *head* and $p_1,\cdots,p_n$ is called a *body*. As an a-term can be separated into an o-term and a set of constraints, the rule can be rewritten as follows:

$$m :: o/|C_H \Leftarrow m_1:o_1,\cdots,m_n:o_n||C_B.$$

where $a \cong o/|C_H$, $p_i \cong m_i : o_i|C_i$, and $C_B = C_1 \cup \cdots \cup C_n$. C_H is a *head constraint* and C_B is a *body constraint*. Their domain is a set of labeled graphs. Note that constraints by a-terms in a body can be included in C_B. Compared with conventional constraint logic programming, a head constraint is new.

A *module* is defined as a set of rules with the same mid. We define the acyclic relation among modules, a *submodule* relation. This works for *rule inheritance* as follows:

$$\begin{aligned} m_1 &\sqsupseteq_S m_2 \\ m_3 &\sqsupseteq_S m_4 \cup (m_5 \setminus m_6) \end{aligned}$$

where m_1 inherits a set of rules in m_2, and m_3 inherits a set of rules defined by set operations such as $m_4 \cup (m_5 \setminus m_6)$. Set operations such as intersection and difference are syntactically evaluated. Even if a module is parametric, that is, the mid is an o-term with variables, the submodule relation can be defined. In order to treat the exception of rule inheritance, each rule has properties such as *local* and *overriding*: a local rule is not inherited to other modules and an overriding rule obstructs the inheritance of rules with the same head from other modules.

A *program* or a *database* is defined as a set of rules with definitions of subsumption and submodule relations. Clearly, a program can be also considered as a set of modules, where an object may have different properties if it exists in different modules. Therefore, we can classify a knowledge-base into different modules and define a submodule relation among them. If a submodule relation is not defined among two modules, even transitively, an object with the same oid may have different (or even inconsistent) properties in its modules. The semantics of a program is defined on the domain of pairs of labeled graphs corresponding to a mid and an o-term. In this framework, we can classify a large-scaled knowledge-base, which might have inconsistencies, and store it in a *QUIXOTE* database.

2.4 Updating and Persistence

QUIXOTE has a concept of nested transaction and allows two kinds of database update:

1) *incremental insert* of a *database* when issuing a query, and

2) *dynamic insert* and *delete* of *o-terms* and *a-terms* during query processing.

We can issue a query with a new database to be added to the existing database. 1) corresponds to the case. For example, consider the following sequence of queries to a database DB:

query sequence to DB		equivalent query
?- **begin_transaction**.		
?- Q_1 **with** DB_1.	$\Longleftrightarrow$	?- Q_1 **to** $DB \cup DB_1$
?- **begin_transaction**.		
?- Q_2 **with** DB_2.	$\Longleftrightarrow$	?- Q_2 **to** $DB \cup DB_1 \cup DB_2$
?- **abort_transaction**.		
?- Q_3 **with** DB_3.	$\Longleftrightarrow$	?- Q_1 **to** $DB \cup DB_1 \cup DB_3$
?- Q_4.	$\Longleftrightarrow$	?- Q_1 **to** $DB \cup DB_1 \cup DB_3$
?- **end_transaction**.		

After successful execution of the above sequence, DB is changed to $DB \cup DB_1 \cup DB_3$. Each DB_i may have definitions of a subsumption relation or a submodule relation, which are merged into definitions of the existing database, If necessary, the subsumption or submodule hierarchy is reconstructed. By rolling back the transaction, such a mechanism can also be used as hypothesis reasoning.

2) makes it possible to update an o-term or its (mutable) properties during query processing, where transactions are located as subtransactions of a transaction in 1). In order to guarantee the semantics of update, so-called AND- and OR-parallel executions are inhibited. For example, the following is a simple rule for updating an employees' salary:

$$
\begin{aligned}
&pay[year=1992, dept=X]/[raise=Y] \\
&\quad \Leftarrow \texttt{begin\_transaction}; \\
&\qquad employee[num=Z]/[dept=X, salary=W]; \\
&\qquad -employee[num=Z]/[salary=W]; \\
&\qquad +employee[num=Z]/[salary=New]; \\
&\qquad \texttt{end\_transaction} \\
&\qquad ||\{New = W * Y\}.
\end{aligned}
$$

where ":" specifies sequential execution in order to suppress AND-parallel execution. "+" means insert, and "−" means delete.

Except for the objects to be deleted or rolled back during query processing, all (extensional or intensional) objects in a *QUIXOTE* program are guaranteed to be persistent. Such persistent objects are stored in the underlying database management system (explained in the next section) or a file system.

2.5 Query Processing and the System

QUIXOTE is basically a constraint logic programming language with object-orientation features such as object identity, complex object, encapsulation, type hierarchy, and methods. However, this query processing is different from conventional query processing because of the existence of oids and head constraints. For example, consider the following program:

$$
\begin{aligned}
&lot[num=X]/[prize_1 \rightarrow a] \Leftarrow X \sqsubseteq 2n. \\
&lot[num=X]/[prize_2 \rightarrow b] \Leftarrow X \sqsubseteq 3n. \\
&lot[num=X]/[prize_1 \rightarrow c] \Leftarrow X \sqsubseteq 5n.
\end{aligned}
$$

where $2n$ is a type with a multiple of two. Given a query $?\text{-}lot[num = 30]/[prize_1 = X, prize_2 = Y]$, the answer is $X \sqsubseteq \text{meet}(a, c)$ and $Y \rightarrow b$, that is,

$$lot[num=30]/[prize_1 \rightarrow \text{meet}(a, c), prize_2 \rightarrow b].$$

First, because of the existence of oids, all rules which possibly have the same oid must be evaluated and merged if necessary. Therefore, in *QUIXOTE*, a query is always processed in order to obtain all solutions. Secondly, as a rule in *QUIXOTE* has two kinds of constraints, a head constraint and a body constraint, each of which consists of equations and inequations of dotted terms besides the variable environment, the derivation process is different from conventional constraint logic programming:

$$(G_0, \emptyset) \rightarrow \cdots \rightarrow (G_i, C_i) \rightarrow \cdots \rightarrow (\emptyset, C_n)$$

where G_i is a set of subgoals and C_i is a set of constraints of the related variables. On the other hand, in *QUIXOTE*, each node in the derivation sequence is (G, A, C), where G is a set of *subgoals*, A is a set of *assumptions* consisting of a body constraint of dotted terms, and C is a set of *conclusions* as a set of constraints consisting of a head constraint and a variable environment. Precisely speaking, the derivation is not a sequence but a directed acyclic graph in

Quixote, because some subsumption relation among assumptions and constraints might force the two sequences to merge: for example, (G, A, C) and (G, A, C') are merged into $(G, A, C \cup C')$. Therefore, the derivation is shown in Figure 4, where the environment to make

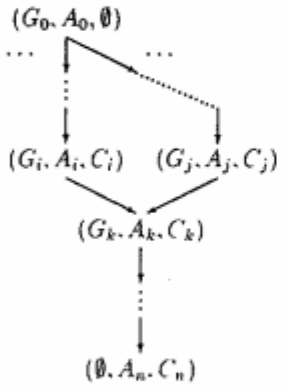


Figure 4: Derivation in *Quixote*

it possible to merge two sequences is restricted: only results by the, so-called, OR-parallel that includes rules inherited by subsumption relation among rule heads can be merged innermostly. The current implementation of query processing in *Quixote* is based on a tabular method such as OLDT in order to obtain all solutions. Sideways information passing is also implemented by considering not only binding information but also property inheritance.

We list some features of the *Quixote* system:

- A *Quixote* program is stored in persistent storage in the form of both the 'source' code and the 'object' code, each of which consists of four parts: control information, subsumption relation, submodule relation, and a set of rules. Persistence is controlled by the *persistence manager*, which switches where programs should be stored. A set of rules in the 'object' code is optimized to separate extensional and intensional databases as in conventional deductive databases.

- When a user builds a huge database in *Quixote*, it can be written as a set of small databases independently of a module concept. These can be gathered into one database, that is, a database can be reused in another database.

- When a user utilizes data and knowledge in *Quixote*, multiple databases can be accessed simultaneously through the *Quixote server*, although the concurrency control of the current version of *Quixote* is simply implemented.

- Users can use databases through their application programs in ESP [Chikayama 1984] or KL1 [Ueda and Chikayama 1990], and through the specific window interface called *Qmacs*.

The environment is shown in Figure 5.

The first version of *Quixote* was released in December, 1991. A second version was released in April, 1992. Both versions are written in KL1 and work on parallel inference machines (PIMs) [Goto *et al.* 1988] and its operating system (PIMOS) [Chikayama *et al.* 1988].

3 Advanced Database Management System (Kappa)

In order to process a large database in *Quixote* efficiently, a database engine called *Kappa* has been developed [5]. In this section, we explain its features.

3.1 Nested Relation and *Quixote*

The problem is which part of *Quixote* should be supported by a database engine because enriched representation is a trade-off in efficient processing. We intend for the database engine to be able to, also, play the role of a practical database management system. Considering the various data and knowledge in our knowledge information processing environment, we adopt an *extended nested relational model*, which corresponds to the class of an o-term without infinite structure in *Quixote*. The term "*extended*" means that it supports a new data type such as Prolog term and provided extensibility as the system architecture for various applications. The reason why we adopt a nested relational model is, not surprisingly, to achieve efficient representation and efficient processing.

Intuitively, a *nested relation* is defined as a subset of a Cartesian product of domains or other nested relations:

$$\begin{aligned} NR &\subseteq E_1 \times \cdots \times E_n \\ E_i &::= D \mid 2^{NR} \end{aligned}$$

where D is a set of atomic values. That is, the relation may have a hierarchical structure and a set of other relations as a value. This corresponds to the introduction of tuple and set constructors. From the viewpoint of syntactical and semantical restrictions, there are various subclasses. Extended relational algebra are defined to each of these.

In Kappa's nested relation, a set constructor is used only as an abbreviation of a set of normal relations as follows:

$$\{r[l_1 = a, l_2 = \{b_1, \cdots, b_n\}]\} \iff \{r[l_1 = a, l_2 = b_1], \cdots, r[l_1 = a, l_2 = b_n]\}$$

[5] See the details in [Kawamura *et al.* 1992].

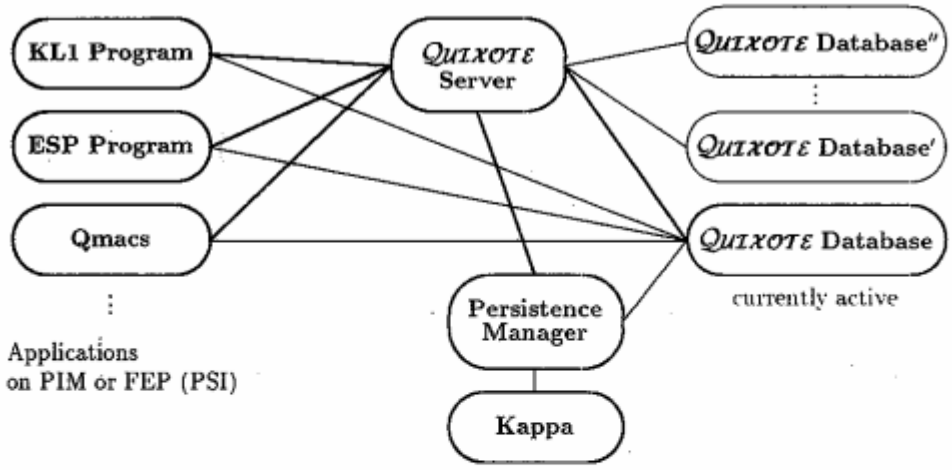


Figure 5: Environment of *QUIXOTE*

The operation of "⇒" corresponds to an *unnest* operation, while the opposite operation ("⇐") corresponds to a *nest* or *group-by* operation, although "⇐" is not necessarily congruent for application of nest or group-by operation sequences, That is, in Kappa, the semantics of a nested relation is the same as the coresponding relation without set constructors. The reason for taking such semantics is to retain first order semantics for efficient processing and to remain compatible to widely used relational databases. Given a nested tuple nt, let the corresponding set of tuples without a set constructor be $\underline{nt}$. Let a nested relation be

$$NR = \{nt_1, \cdots, nt_n\}$$
$$\text{where } \underline{nt_i} = \{t_{i1}, \cdots, t_{ik}\} \text{ for } i = 1, \cdots, n,$$

then the semantics of NR is

$$\bigcup_{i=1}^{n} \underline{nt_i} = \{t_{11}, \cdots, t_{1k}, \cdots, t_{n1}, \cdots, t_{nk}\}.$$

Extended relational algebra to this nested relational database is defined in Kappa and produces results according to the above semantics, which guarantees to produce the same result to the corresponding relational database, except for treatment of the label hierarchy.

A query can be formulated as a first order language. we, generally, consider this in the form of a rule constructed by nested tuples. As the relation among facts in a database is conjunctive from a proof-theoretic point of view, the semantics of a rule is clear according to the above semantics. For example, the following rule

$$r[l_1 = X, l_2 = \{a, b, c\}]$$
$$\Leftarrow B, r'[l_2 = Y, l_3 = \{d, e\}, l_3 = Z], B'.$$

can be transformed into the following set of rules without set constructors:

$$r[l_1 = X, l_2 = a]$$
$$\Leftarrow B, r'[l_2 = Y, l_3 = d, l_3 = Z], r'[l_2 = Y, l_3 = e, l_3 = Z], B'.$$
$$r[l_1 = X, l_2 = b]$$
$$\Leftarrow B, r'[l_2 = Y, l_3 = d, l_3 = Z], r'[l_2 = Y, l_3 = \epsilon, l_3 = Z], B'.$$
$$r[l_1 = X, l_2 = c]$$
$$\Leftarrow B, r'[l_2 = Y, l_3 = d, l_3 = Z], r'[l_2 = Y, l_3 = \epsilon, l_3 = Z], B'.$$

That is. each rule can also be unnested. The point of efficiently processing Kappa relations is to reduce the number of unnest and nest operations: that is. to process sets as directly as possible.

Under the semantics. query processing to nested relations is different from conventional procedures in logic programming. For example. consider a simple database consisting of only one tuple:

$$r[l_1 = \{a, b\}, l_2 = \{b, c\}].$$

For a query $?\text{-}r[l_1 = X, l_2 = X]$, we can get $X = \{b\}$. that is. an intersection of $\{a, b\}$ and $\{b, c\}$. That is. a concept of unification should be extended. In order to generalize such a procedure, we must introduce two concepts into the procedural semantics[Yokota 1988]:

1) *Residue Goals*
Consider the following program and a query:

$$r[l = S'] \Leftarrow B.$$
$$?\text{-}r[l = S].$$

If $S \cap S'$ is not an empty set during unification between $r[l = S]$ and $r[l = S']$. new subgoals are to be $r[l = S \setminus S'], B$. That is. a residue subgoal $r[l = S \setminus S']$ is generated if $S_1 \setminus S_2$ is not an empty set. otherwise the unification fails. Note that there might be residue subgoals if there are multiple set values.

2) *Binding as Constraint*
Consider the following database and a query:

$$r_1[l_1 = S_1].$$

$r_2[l_2 = S_2]$.
?-$r_1[l_1 = X], r_2[l_2 = X]$.

Although we can get $X = S_1$ by unification between $r_1[l_1 = X]$ and $r_1[l_1 = S_1]$ and a new subgoal $r_2[l_2 = S_1]$, the subsequent unification results in $r_2[l_2 = S_1 \cap S_2]$ and a residue subgoal $r_2[l_2 = S_1 \setminus S_2]$. Such a procedure is wrong, because we should have an answer $X = S_1 \cap S_2$. In order to avoid this situation, the binding information is temporary and plays the role of constraints to be retained:

$$r_1[l_1 = X], r_2[l_2 = X]$$
$$\Longrightarrow r_2[l_2 = X] || \{X \subset S_1\}$$
$$\Longrightarrow || \{X \subset S_1 \cap S_2\}.$$

There remains one problem where the unique representation of a nested relation is not necessarily decided in the Kappa model, as already mentioned. In order to decide a unique representation, each nested relation has a sequence of labels to be nested in Kappa.

As the procedural semantics of extended relational algebra in Kappa is defined by the above concepts, a Kappa database does not necessarily have to be *normalized* also in the sense of nested relational models, in principle. That is, it is unnecessary for users to be conscious of the row nest structure.

Furthermore, nested relational model is well known to reduce the number of relations in the case of multivalue dependency. Therefore, the Kappa model guarantees more efficient processing by reducing the number of tuples and relations, and more efficient representation by complex construction than the relational model.

3.2 Features of Kappa System

The nested relational model in Kappa has been implemented. This consists of a sequential database management system *Kappa-II* [Yokota *et al.* 1988] and a parallel database management system *Kappa-P* [Kawamura *et al.* 1992]. Kappa-II, written in ESP, works on sequential inference machines (PSIs) and its operating system (SIMPOS). Kappa-P, written in KL1, works on parallel inference machines (PIMs) and its operating system (PIMOS). Although their architectures are not necessarily the same because of environmental differences, we explain their common features in this subsection.

- *Data Type*
  As Kappa aims at a database management system (DBMS) in a knowledge information processing environment, a new data type, *term*, is added. This is because various data and knowledge are frequently represented in the form of terms. Unification and matching are added for their operations. Although unification-based relational algebra can emulate the derivation in logic programming, the features are not supported in Kappa because the algebra is not so efficient. Furthermore, Kappa discriminates one-byte character (ASCII) data from two-byte character (JIS) data as data types. It contributes to the compression of huge amounts of data such as genetic sequence data.

- *Command Interfaces*
  Kappa provides two kinds of command interface: *basic commands* as the low level interface and extended relational algebra as the high level interface. In many applications, the level of extended relational algebra, which is expensive, is not always necessary. In such applications, users can reduce the processing cost by using basic commands.

  In order to reduce the communication cost between a DBMS and a user program, Kappa provides user-definable commands, which can be executed in the same process of the Kappa kernel (in Kappa-II) or the same node of each local DBMS (in Kappa-P, to be described in the next subsection).

  The user-definable command facility helps users design any command interface appropriate for their application and makes their programs run efficiently. Kappa's extended relational algebra is implemented as parts of such commands although it is a built-in interface.

- *Practical Use*
  As already mentioned, Kappa aims, not only at a database engine of QUIXOTE, but also at a practical DBMS, which works independently of QUIXOTE. To achieve this objective, there are several extensions and facilities. First, new data types, besides the data types mentioned above, are introduced in order to store the environment under which applications work. There are *list*, *bag*, and *pool*. They are not, however, supported fully in extended relational algebra because of semantic difficulties.

  Kappa supports the same interface to such data types as in SIMPOS or PIMOS.

  In order to use Kappa databases from windows, Kappa provides a user-friendly interface, like a spreadsheet, which provides an ad hoc query facility including update, a browsing facility with various output formats and a customizing facility.

- *Main Memory Database*
  Frequently accessed data can be loaded and re-

tained in the main memory as a main memory database. As such a main memory database was designed only for efficient processing of temporary relations without additional burdens in Kappa, the current implementation does not support conventional mechanisms such as deferred update and synchronization. In Kappa-P, data in a main memory database are processed at least three times more efficiently than in a secondary storage database.

From an implementational point of view, there are several points for efficient processing in Kappa. We explain two of them:

- *ID Structure and Set Operation*
  Each nested tuple has a unique tuple identifier (*ntid*) in a relation, which is treated as an 'object' to be operated explicitly. Abstractly speaking, there are four kinds of 'object's, such as a *nested tuple*, an *ntid*, a *set* whose element is a ntid, and a *relation* whose element is a nested tuple. Their commands for transformation are basically supported, as in Figure 6, although the set

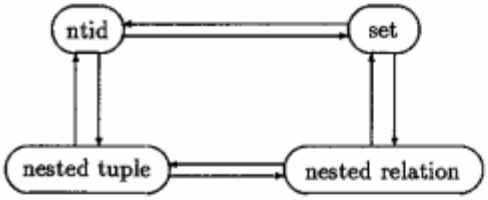


Figure 6: 'Object's in Kappa and Basic Operations

is treated as a *stream* in Kappa-P. Most operations are processed in the form of an ntid or a set.

In order to process a selection result, each subtuple in a nested tuple also has a *sub-ntid* virtually. Set operations (including unnest and nest operation) are processed mainly in the form of a (sub-)ntid or a set without reading the corresponding tuples.

- *Storage Structure*
  A nested tuple, which consists of unnested tuples in the semantics, is also considered as a set of unnested tuples to be accessed together. So, a nested tuple is compressed *without decomposition* and stored on the same page, in principle, in the secondary storage. For a huge tuple, such as a genetic sequence, contiguous pages are used. In order to access a tuple efficiently, there are two considerations: how to locate the necessary tuple efficiently, and how to extract the necessary attributes efficiently from the tuple. As in Figure 7.

Kappa is equipped with an efficient address translation table between an ntid and a logical page (*lp*), and between a logical page and a physical page (*pp*). This table is used by the underlying file system. For extraction purposes, each node of

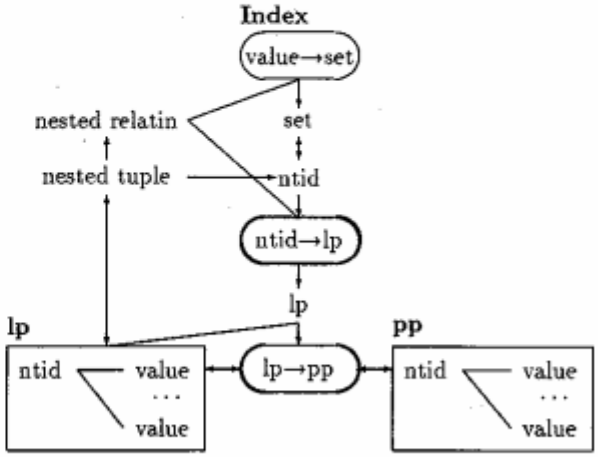


Figure 7: Access Network for Secondary DBMS

a nested tuple has a local pointer and counters in the compressed tuple, although there is a trade-off in update operations' efficiency.

Each entry in an index reflects the nested structure: that is, it contains any necessary sub-ntids. The value in the entry can be the result of string operations such as substring and concatenation of the original values, or a result extracted by a user's program.

3.3 Parallel Database Management System (Kappa-P)

Kappa-P has various unique features as a parallel DBMS. In this subsection, we give a brief overview of them.

The overall configuration of Kappa-P is shown in Figure 8. There are three components: an *interface (I/F) process*, a *server DBMS*, and a *local DBMS*. An I/F process, dynamically created by a user program, mediates between a user program and (server or local) DBMSs by *streams*. A server DBMS has a global map of the location of local DBMSs and makes a user's stream connect directly to an appropriate local DBMS (or multiple local DBMSs). In order to avoid a bottleneck in communication, there might be many server DBMSs with replicates global maps. A local DBMS can be considered as a single nested relational DBMS, corresponding to Kappa-II, where users' data is stored.

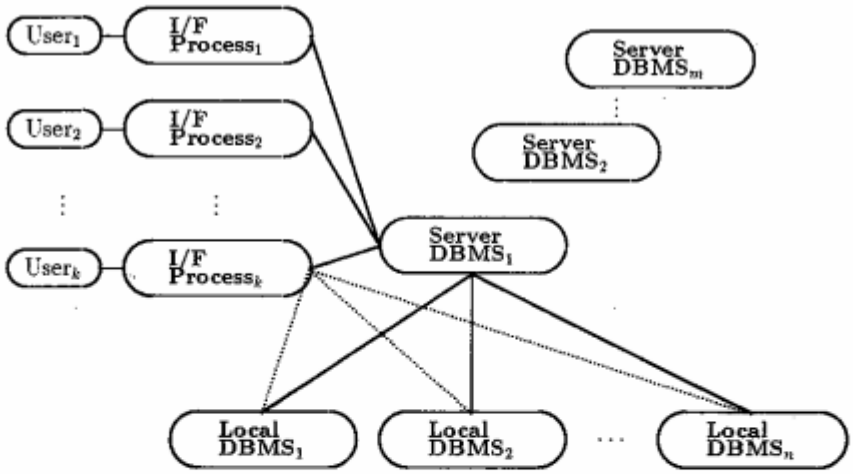


Figure 8: Configuration of Kappa-P

Users' data may be distributed (even horizontally partitioned) or replicated into multiple local DBMSs. If each local DBMS is put in a shared memory parallel processor, called a *cluster* in PIM, each local DBMS works in parallel. Multiple local DBMSs are located in each node of distributed memory parallel machine, and, together, behave like a distributed DBMS.

User's procedures using extended relational algebra are transformed into procedures written in an intermediate language, the syntax of which is similar to KL1, by an interface process. During the transformation, the interface process decides which local DBMS should be the coordinator for the processing, if necessary. Each procedure is sent to the corresponding local DBMS, and processed there. Results are gathered in the coordinator and then processed.

Kappa-P is different from most parallel DBMS, in that most users' applications also work in the same parallel inference machine. If Kappa-P coordinates a result from results obtained from local DBMSs, as in conventional distributed DBMSs, even when such coordination is unnecessary, the advantages of parallel processing are reduced. In order to avoid such a situation, the related processes in a user's application can be dispatched to the same node as the related local DBMS as in Figure 9. This function contributes not only to efficient processing but also to customization of the command interface besides the user-defined command facility.

4 Applications

We are developing three applications on *Quixote* and Kappa, and give an overview of each research topic in this section.

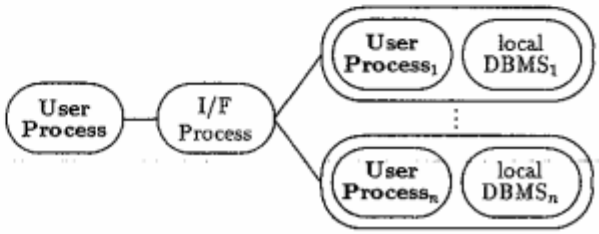


Figure 9: User's Process in Kappa Node

4.1 Molecular Biological Database

Genetic information processing systems are very important not only from scientific and engineering points of view but also from a social point of view, as shown in the Human Genome Project. Also, at ICOT, we are engaged in such systems from thr viewpoint of knowledge information processing. In this subsection, we explain such activities, mainly focusing on molecular biological databases in *Quixote* and Kappa [6].

4.1.1 Requirements for Molecular Biological Databases

Although the main objective of genetic information processing is to design proteins as the target and to produce them, there remain too many technical difficulties presently. Considering the whole of proteins, we are only just able to gather data and knowledge with much noise.

In such data and knowledge there are varieties such as sequences, structures, and functions of genes and proteins, which are mutually related. A gene in the

[6] See the details in [Tanaka 1992].

genetic sequence (DNA) in the form of a *double helix* is copied to a mRNA and translated into an amino acid sequence, which becomes a part (or a whole) of a protein. Such processes are called the Central Dogma in biology. There might be different amino acids even with the same functions of a protein. The size of a unit of genetic sequence data ranges from a few characters to around 200 thousand. and will become longer as genome data is gradually analyzed fyrther. The size of a human genome sequence equals about 3 billion characters. As there are too many unknown proteins. the sequence data is fundamental for homology searching by a pattern called a motif and for multiple alignment among sequences for prediction of the functions of unknown proteins from known ones.

There are some problems to be considered for molecular biological databases:

- how to store large values, such as sequences. and process them efficiently.
- how to represent structure data and what operations to apply them,
- how to represent functions of protein such as chemical reactions, and
- how to represent their relations and link them.

From a database point of view, we should consider some points in regard to the above data and knowledge:

- representation of complex data as in Figure 2.
- treatment of partial or noisy information in unstable data,
- inference rules representing functions, as in the above third item, and inference mechanisms, and
- representation of hierarchies such as biological concepts and molecular evolution.

After considering the above problems, we choose to build such databases on a DOOD (*Quixote*, conceptually), while a large amount of simple data is stored in Kappa-P and directly operated through an optimized window interface, for efficient processing. As cooperation with biologists is indispensable in this area, we also implemented an environment to support them. The overall configuration of the current implementation is shown in Figure 10.

4.1.2 Molecular Biological Information in *Quixote* and Kappa

Here, we consider two kinds of data as examples: sequence data and protein function data.

First, consider a DNA sequence. Such data does not need inference rules. but needs a strong capability for homology searching. In our system. such data is stored

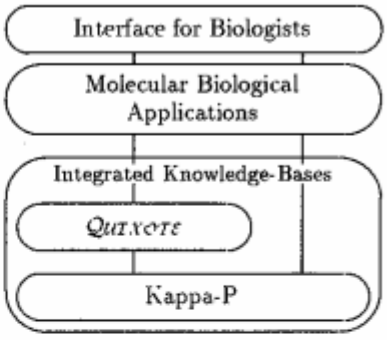


Figure 10: Integrated System on *Quixote* and Kappa

directly in Kappa. which supports the storage of much data as is and creates indexes from the substrings extracted from the original by a user program. Sequence-oriented commands for information retrieval. which use such indexes. can be embedded into Kappa as user-defined commands. Furthermore. since the complex record shown in Figure 3 is treated like a nested relation. the representation is also efficient. Kappa shows its effectiveness as a practical DBMS.

Secondly. consider a chemical reaction of enzymes and co-enzymes. whose scheme is as follows:

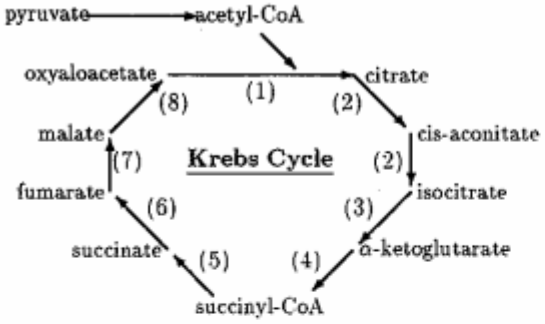
$$\text{Sources} + \text{Co-enzymes} \xrightarrow[\text{Environments}]{\text{Enzymes}} \text{Products}$$

As an example of metabolic reaction. consider the Krebs cycle in Figure 11. Chemical reactions in the Krebs cycle are written as a set of facts in *Quixote* as in Figure 12. In the figure, $o_1 \sqsubseteq o_2/[\cdots]$ means $o_1/[\cdots]$ and $o_1 \sqsubseteq o_2$. In order to obtain a reaction chain (path) from the above facts. we can write the following rules in *Quixote*:

$$\begin{aligned} &reaction[from = X, to = Y] \\ &\quad \Leftarrow W \sqsubseteq reaction/[sources^{+} \leftarrow X, \\ &\qquad\qquad products^{+} \leftarrow Z]. \\ &\qquad reaction[from = X, to = Y] \\ &\qquad \|\{\{X, Y, Z\} \sqsubseteq reaction\}. \\ &reaction[from = X, to = X] \\ &\quad \Leftarrow \|\{X \sqsubseteq reaction\}. \end{aligned}$$

Although there are a lot of difficulties in representing such functions. *Quixote* makes it possible to write them down easily.

Another problem is how to integrate a Kappa database with a *Quixote* database. Although one of the easiest ways is to embed the Kappa interface into *Quixote*. it costs more and might destroy a uniform representation in *Quixote*. A better way would be to manage common oids both in Kappa and in *Quixote*. and guarantee the common object. however we have



ENZYMES

(1) citrate synthase
(2) aconitate
(3) isocitrate dehydrogenase
(4) α-ketoglutarate dehydrogenase complex
(5) succinyl-CoA synthetase
(6) succinate dehydrogenase
(7) fumarase
(8) malate dehydrogenas

Figure 11: Krebs Cycle in Metabolic Reaction

not implemented such a facility in Kappa. The current implementation puts the burden of the uniformity on the user. as in Figure 10.

4.2 Legal Reasoning System (TRIAL)

Recently, legal reasoning has attracted much attention from researchers in artificial intelligence. with high expectations for its *big* application. Some prototype systems have been developed. We also developed such a system as one of the applications of our DOOD system [7].

4.2.1 Requirements for Legal Reasoning Systems and TRIAL

First. we explain the features of legal reasoning. The analytical legal reasoning process is considered as consisting of three steps: *fact findings. statutory interpretation*, and *statutory application*.

Although fact findings is very important as the starting point. it is too difficult for current technologies. So. we assume that new cases are already represented in the appropriate form for our system. Statutory interpretation is one of the most interesting themes from an artificial intelligence point of view. Our legal reasoning system. TRIAL. focuses on statutory interpretation as well as statutory application.

[7]See the details in [Yamamoto 1990]. although the new version is revised as in this section.

```
krebs_cycle :: {{
    krebs1 ⊑ reaction/
             [sources+ ← {acetylcoa, oxaloacetate}.
              products+ ← {citrate, coa}.
              enzymes ← citrate_synthase.
              energy = −7.7].
    krebs2 ⊑ reaction/
             [sources+ ← citrate.
              products+ ← {isocitrate, h2o}.
              enzymes ← aconitase].
      ⋮
    krebs8 ⊑ reaction/
             [sources+ ← malate.
              products+ ← oxaloacetate.
              enzymes ← malate_dehydrogenase.
              energy = 7.1].
             }}
```

Figure 12: Facts of Krebs Cycle in *Quixote*

Although there are many approaches to statutory interpretation. we take the following steps:

- *analogy detection*
  Given a new case. similar precedents to the case are retrieved from an existing precedent database.
- *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 abstract interpretation rules transformed by rule transformation. This step may include statutory application because it is used in the same manner.

Among the steps. the strategy for analogy detection is essential in legal reasoning for *more efficient* detection of *better* precedents. which decides the quality of the results of legal reasoning. As the primary objective of TRIAL at the current stage is to investigate the possibilities of *Quixote* in the area and develop a prototype system. we focus only on a small target. That is. to what extent should interpretation rules be abstracted for *a* new case. in order to get an answer with a plausible explanation. but not for general abstraction mechanism.

4.2.2 TRIAL on Legal Precedent Databases

All data and knowledge in TRIAL is described in *Quixote*. The system. written in KL1. is constructed on *Quixote*. The overall architecture is shown in Figure 13. In the figure. *Quixote* supports the functions of rule transformation (Rule Transformer) and deductive reasoning (Deductive Reasoner) as the native functions besides the database component. while TRIAL

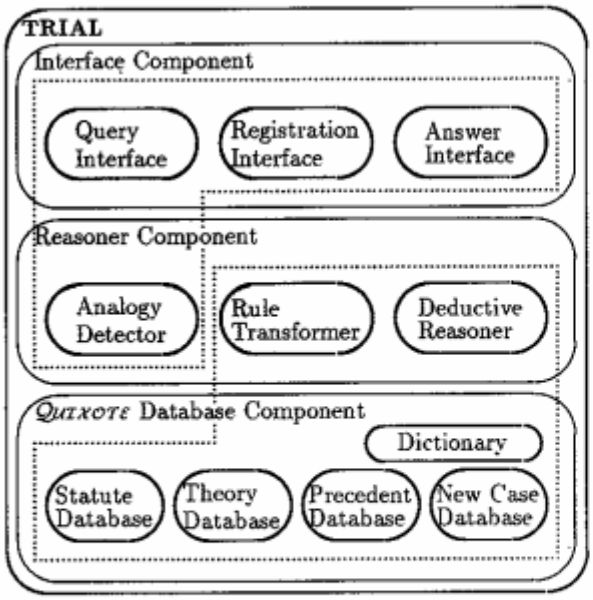


Figure 13: Architecture of TRIAL

supports the function of analogy detection (Analogy Detector) besides the interface component.

Consider a simplified example related to "*karoshi*" (death from overwork) in order to discuss the analogy detector. A new case, *new-case*, is as follows:

> *Mary*, a *driver*, employed by a company, "*S*", died from a *heart-attack* while taking a *catnap* between jobs. Can this case be applied to the *worker's compensation law*?

This is represented as a module *new-case* in QUIXOTE as follows:

$$
\begin{aligned}
\mathit{new\text{-}case} :: \{\{\mathit{new\text{-}case}/[&\mathit{who} = \mathit{mary}.\\
&\mathit{while} = \mathit{catnap}.\\
&\mathit{result} = \mathit{heart\text{-}attack}]::\\
\mathit{rel}[&\mathit{state} = \mathit{employee}, \mathit{emp} = \mathit{mary}]\\
&/[\mathit{affil} = \mathit{org}[\mathit{name} = \text{"}S\text{"}].\\
&\mathit{job} \rightarrow \mathit{driver}]\}\}
\end{aligned}
$$

where ";;" is a delimiter between rules. The module is stored in the new case database. Assume that there are two *abstract* precedents [8] of *job-causality* and *job-execution*:

$$
\begin{aligned}
\mathit{case}_1 :: \mathit{judge}&[\mathit{case} = X]/[\mathit{judge} \rightarrow \mathit{job\text{-}causality}]\\
&\Leftarrow \mathit{rel}[\mathit{state} = Y, \mathit{emp} = Z]/[\mathit{cause} = X]\\
&\quad \|\{X \sqsubseteq \mathtt{parm}.\mathit{case}.\\
&\qquad Y \sqsubseteq \mathtt{parm}.\mathit{status}.\\
&\qquad Z \sqsubseteq \mathtt{parm}.\mathit{emp}\}::\\
\mathit{case}_2 :: \mathit{judge}&[\mathit{case} = X]/[\mathit{judge} \rightarrow \mathit{job\text{-}execution}]\\
&\Leftarrow X/[\mathit{while} = Y, \mathit{result} = Z].\\
&\quad Y \sqsubseteq \mathit{job}\\
&\quad \|\{X \sqsubseteq \mathtt{parm}.\mathit{case}.\\
&\qquad Y \sqsubseteq \mathtt{parm}.\mathit{while}.\\
&\qquad Z \sqsubseteq \mathtt{parm}.\mathit{result}\}.
\end{aligned}
$$

Note that variables X, Y, and Z in both rules are restricted by the properties of an object **parm**. That is, they are already abstracted by **parm** and their abstract level is controlled by **parm**'s properties. Such precedents are retrieved from the precedent database by analogy detection and abstracted by rule transformation. We must consider the *labor-law* (in the statute database) and a *theory* (in the theory database) as follows:

$$
\begin{aligned}
\mathit{labor\text{-}law} :: \mathit{org}&[\mathit{name} = X]\\
&/[\mathit{resp} \rightarrow \mathit{compensation}[\mathit{obj} = Y.\\
&\qquad\qquad \mathit{money} = \mathit{salary}]]\\
&\Leftarrow \mathit{judge}[\mathit{case} \rightarrow \mathit{case}]\\
&\qquad /[\mathit{who} = Y.\\
&\qquad\quad \mathit{result} \rightarrow \mathit{disease},\\
&\qquad\quad \mathit{judge} \rightarrow \mathit{insurance}].\\
&\quad \mathit{rel}[\mathit{state} = Z, \mathit{emp} = Y]\\
&\qquad /[\mathit{affil} = \mathit{org}[\mathit{name} = X]].
\end{aligned}
$$

$$
\begin{aligned}
\mathit{theory} :: \mathit{judge}&[\mathit{case} = X]/[\mathit{judge} \rightarrow \mathit{insurance}]\\
&\Leftarrow \mathit{judge}[\mathit{case} = X]/[\mathit{judge} \rightarrow \mathit{job\text{-}causality}].\\
&\quad \mathit{judge}[\mathit{case} = X]/[\mathit{judge} \rightarrow \mathit{job\text{-}execution}]\\
&\quad \|\{X \sqsubseteq \mathit{case}\}.
\end{aligned}
$$

Furthermore, we must define the **parm** object as follows:

$$
\begin{aligned}
\mathit{parm} :: \mathtt{parm}/[&\mathit{case} = \mathit{case}.\\
&\mathit{state} = \mathit{rel}.\\
&\mathit{while} = \mathit{job}.\\
&\mathit{result} = \mathit{disease}.\\
&\mathit{emp} = \mathit{person}].
\end{aligned}
$$

In order to use **parm** for case_1 and case_2, we define the following submodule relation:

$$\mathit{parm} \sqsupseteq_S \mathit{case}_1 \cup \mathit{case}_2.$$

This information is dynamically defined during rule transformation. Furthermore, we must define the subsumption relation:

$$
\begin{array}{lcl}
\mathit{case} & \sqsupseteq & \mathit{new\text{-}case}\\
\mathit{rel} & \sqsupseteq & \mathit{employee}\\
\mathit{disease} & \sqsupseteq & \mathit{heart\text{-}attack}\\
\mathit{job} & \sqsupseteq & \mathit{catnap}\\
\mathit{person} & \sqsupseteq & \mathit{mary}\\
\mathit{job\text{-}causality} & \sqsupseteq & \mathit{insurance}\\
\mathit{job\text{-}execution} & \sqsupseteq & \mathit{insurance}
\end{array}
$$

[8] In this paper, we omit the rule transformation step and assume that abstract interpretation rules are given.

Such definitions are stored in the dictionary in advance.

Then, we can ask some questions with a hypothesis to the above database:

1) If *new-case* inherits *parm* and *theory*, then what kind of *judgment* can we get?

$$?\text{-}new\text{-}case : judge[case = new\text{-}case]/[judge = X]$$
$$\texttt{if } new\text{-}case \sqsupseteq_S parm \cup theory.$$

we can get three answers:

- $X = job\text{-}execution$
- if $new\text{-}case : judge[case = new\text{-}case]$ has a property $judge \sqsubseteq job\text{-}causality$, then $X \sqsubseteq insurance$
- if $new\text{-}case : rel[state = employee, emp = mary]$ has a property $cause = new\text{-}case$, then $X \sqsubseteq insurance$

Two of these are answers with assumptions.

2) If *new-case* inherits *labor-law* and *parm*, then what kind of responsibility should the organization which Mary is affiliated to have?

$$?\text{-}new\text{-}case : org[name = \text{"}S\text{"}]/[resp = X]$$
$$\texttt{if } new\text{-}case \sqsupseteq_S parm \cup labor\text{-}law.$$

we can get two answers:

- if $new\text{-}case : judge[case = new\text{-}case]$ has a property $judge \sqsubseteq job\text{-}causality$, then $X \sqsubseteq compensation[obj = mary, money = salary]$
- if $new\text{-}case : rel[state = employee, emp = mary]$ has a property $cause = new\text{-}case$, then $X \sqsubseteq compensation[obj = mary, money = salary]$

For analogy detection, the **parm** object plays an essential role in determining how to abstract rules as in $case_1$ and $case_2$, what properties to be abstracted in **parm**, and what values to be set in properties of **parm**. In TRIAL, we have experimented with such abstraction, that is, analogy detection, in *Quixote*.

For the user interface of TRIAL, *Quixote* returns explanations (derivation graphs) with corresponding answers, if necessary. The TRIAL interface shows this graphically according to the user's request. By judging an answer from the validity of the assumptions and the corresponding explanation, the user can update the database or change the abstraction strategy.

4.3 Temporal Inference

Temporal information plays an important role in natural language processing. A time axis in natural language is, however, not homogeneous as in natural science but is relative to the events in mind: shrunken in parts and stretched in others. Furthermore, the relativity is different depending on the observer's perspective. This work aims to show the paradigm of an inference system that merges temporal information extracted from each lexical item and resolves any temporal ambiguity that a word may have [9].

4.3.1 Temporal Information in Natural Language

We can, frequently, make different expressions for the same real situation. For example,

Don Quixote attacks a windmill.
Don Quixote attacked a windmill.
Don Quixote is attacking a windmill.
⋮

Such different expressions are related to tense and aspects. How should we describe the relation between them?

According to situation theory, we write a *support* relation between a *situation* s and an *infon* σ as follows:

$$s \models \sigma.$$

For example, if one of the above examples is supported in a situation s, it is written as follows:

$$s \models \ll attack, \text{Don Quixote}, windmill \gg,$$

where *attack* is a relation, and "Don Quixote" and *windmill* are parameters. However, strictly speaking, as such a relation is cut out from a prespective $\mathcal{P}$, we should write it as follows:

$$s \models \sigma \iff \mathcal{P}(s' \models \sigma').$$

Although we might nest perspectives on such a relation, we assume some reflective property:

$$\mathcal{P}(s' \models \sigma') \implies \mathcal{P}(s')\mathcal{P}(\models)\mathcal{P}(\sigma').$$

In order to consider how to represent $\mathcal{P}(s')$ and $\mathcal{P}(\sigma')$ from a temporal point of view, we introduce a partial order relation among sets of time points. Assume that a set of time points are partially ordered by $\preceq$, then we can define $\preceq_t$ and $\subseteq$ among sets T_1 and T_2 as follows:

$$T_1 \preceq_t T_2 \stackrel{def}{=} \forall t_1 \in T_1. \forall t_2 \in T_2. t_1 \preceq t_2.$$
$$T_1 \subseteq T_2 \stackrel{def}{=} \forall t_1 \in T_1. t_1 \in T_2.$$

We omit the subscript t if there is no confusion.

In order to make tense and aspects clearer, we introduce the following concepts:

[9] See the details in [Tojo and Yasukawa 1992].

1) discrimination of an *utterance situation* u and a *described situation* s, and

2) *duration* (a set of linear time points, decided by a start point and an end point) of situations and an infon. The duration of T is written as $\| T \|_t$.

We can see the relation among three durations of an utterance situation, a described situation, and an infon in Figure 14. If there is no confusion, we use a simple

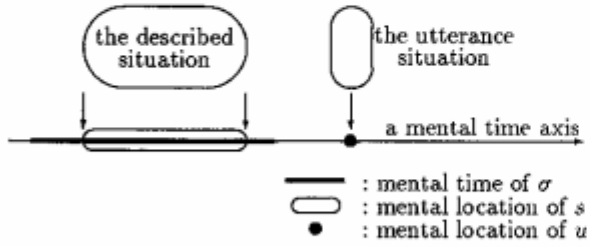


Figure 14: Relation of Three Durations

notation: $s_1 \preceq s_2$ instead of $\| s_1 \|_t \preceq \| s_2 \|_t$ and $s_1 \subseteq s_2$ instead of $\| s_1 \|_t \subseteq \| s_2 \|_t$.

By the above definitions, we can define tense and aspects when $s \models \sigma$ as follows ($\mathcal{P}(\models)$ is written as $\models$):

$$\begin{aligned} s[s \prec u] &\models \ll past, \sigma \gg . \\ s[s \supset u] &\models \ll present, \sigma \gg . \\ s[s \subset u] &\models \ll progressive, \sigma \gg . \\ s[\sigma \prec u] &\models \ll perfect, \sigma \gg . \end{aligned}$$

where s is a described situation, u is an utterance situation, and σ is an infon. C in $s[C]$ is a constraint, which is intended to be a perspective. The above rules are built-in rules (or axioms) for temporal inference in *Quixote*.

4.3.2 Temporal Inference in *Quixote*

We define a rule for situated inference as follows:

$$s \models \sigma \Leftarrow s_1 \models \sigma_1, \cdots, s_n \models \sigma_n.$$

where $s, s_1, \cdots, s_n$ are situations with perspectives. This rule means that $s \models \sigma$ if $s_1 \models \sigma_1, \cdots$, and $s_n \models \sigma_n$. Such rules can be easily translated into a subclass of *Quixote* by relating a situation with perspectives to a module, an infon to an o-term, and partial order among duration to subsumption relation. However, there is one restriction: a constraint in a rule head may not include subsumption relations between o-terms, because such a relation might destroy a subsumption lattice.

A verbalized infon is represented as an o-term as follows [10]:

$$\begin{aligned} inf[v\_rel = [rel = R, \\ cls = CLS, \\ per = P], \\ args = Args], \end{aligned}$$

where v_rel takes a verb relation and $args$ takes the arguments. R is a verb, CLS is the classification, and P is a temporal situation. For example, "John is running" is written as follows:

$$\begin{aligned} inf[v\_rel = [rel = run, \\ cls = act_2, \\ pers = [fov = ip, \\ pov = pres]]. \\ args = [agt = john]]. \end{aligned}$$

That is, the agent is *john*, and the verb is *run*, which is classified in act_2 (*in-progress state* or *resultant state*), and the perspective is *in-progress state* as the field of view (an oval in Figure 14) and *present* as the point of view ($\bullet$ in Figure 14).

The discourse situation which supports such a verbalized infon is represented as follows:

$$dsit[fov = ip, pov = pres, src = U].$$

where the first two arguments are the same as the above infon's *pers* and the third argument is the utterance situation.

According to the translation, we show a small example, which makes it possible to reduce temporal ambiguity in expression.

1) Given an expression $exp = E$, each morpheme is processed in order to check the temporal information:

$$\begin{aligned} &mi[u = U, exp = [\,], \epsilon = D, infon = Infon]. \\ &mi[u = U, exp = [Exp|R], \epsilon = D, infon = Infon] \\ &\quad \Leftarrow d\_cont[exp = Exp, sit = D, infon = Infon, \\ &\qquad mi[u = U, exp = R, \epsilon = D, infon = Infon]. \end{aligned}$$

Temporal information for each morpheme is intersected in D: that is, ambiguity is gradually reduced.

2) Temporal information in a pair of a discourse situation and a verbalized infon is defined by the following rules:

[10] An o-term $T[l_1 = o_1, \cdots, l_n = o_2]$ can be abbreviated as $[l_1 = o_1, \cdots, l_n = o_2]$.

$d\_cont[exp=Exp,$
$\quad sit=dsit[fov=Fov, pov=Pov, src=U]$
$\quad infon=inf[v\_rel=V\_rel, args=Args]]$
$\Leftarrow dict : v[cls=CLS, rel=R, form=Exp]$
$\quad ||\{V\_rel=[rel=R, cls=CLS, pers=P]\}$

$d\_cont[exp=Exp,$
$\quad sit=dsit[fov=Fov, pov=Pov, src=U]$
$\quad infon=inf[v\_rel=V\_rel, args=Args]]$
$\Leftarrow dict : auxv[asp=ASP, form=Exp].$
$\quad map[cls=CLS.asp=ASP, fov=Fov]$
$\quad ||\{V\_rel=[rel=\_.cls=CLS, pers=P].$
$\quad P=[fov=Fov, pov=\_\}];;$

$d\_cont[exp=Exp,$
$\quad sit=dsit[fov=Fov, pov=Pov, src=U]$
$\quad infon=inf[v\_rel=V\_rel.args=Args]]$
$\Leftarrow dict : affix[pov=Pov, form=\mathtt{ru}]$
$\quad ||\{V\_rel=[rel=\_.cls=\_.pers=P],$
$\quad P=[fov=\_.pov=Pov]\}$

3) There is a module *dict*, where lexical information is defined as follows:

$dict::\{\{$
$\quad v[cls=act_1, rel=put\_on, form=\mathtt{ki}];;$
$\quad v[cls=act_2, rel=run, form=\mathtt{hashi}];;$
$\quad v[cls=act_3, rel=understand, form=\mathtt{waka}];;$
$\quad auxv[asp=state, form=\mathtt{tei}];,$
$\quad affix[pov=pres, form=\mathtt{ru}];;$
$\quad affix[pov=past, form=\mathtt{ru}]\}\}$

where *form* has a value of Japanese expression. Further, mapping of field of view is also defined as a set of (global) facts as follows:

$map[cls=act1.asp=state.fov=\{ip.tar.res\}].$
$map[cls=act2.asp=state.fov=\{ip.res\}].$
$map[cls=act3.asp=state.fov=\{tar.res\}].$

If some Japanese expression is given in a query, the corresponding temporal information is returned by the above program.

5 Towards More Flexible Systems

In order to extend a DOOD system, we take other approaches for more flexible execution control, mainly focusing on natural language applications as its examples.

5.1 Constraint Transformation

There are many natural language grammar theories: transformational and constraint-base grammar such as GB, unification-based and rule-based grammar such as GPSG and LFG, and unification-based and constraint-based grammar such as HPSG and JPSG. Considering a more general framework of grammar in logic programming, HPSG and JPSG are considered to be better, because morphology, syntax, semantics, and pragmatics are uniformly treated as constraints. From such a point of view, we developed a new constraint logic programming (CLP) language, *cu-Prolog*, and implemented a JPSG (Japanese Phrase Structure Grammar) parser in it [11].

5.1.1 Constraints in Unification-Based Grammar

First, consider various types of constraints in constraint-based grammar:

- A *disjunctive feature structure* is used as a basic information structure, defined like nested tuples or complex objects as follows:

  1) A *feature structure* is a tuple consisting of pairs of a label and a value:
  $[l_1=v_1, \cdots, l_n=v_n]$.

  2) A *value* is an atom, a feature structure, or a set $\{f_1, \cdots, f_n\}$ of feature structures.

- In JPSG, grammar rules are described in the form of a binary tree as in Figure 15, each node of which is a feature structure: in which a specific

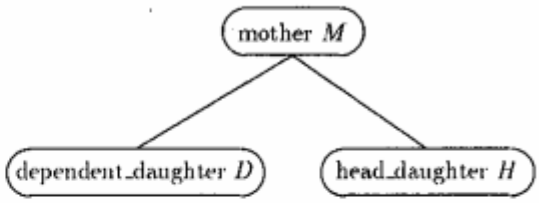


Figure 15: Phrase Structure in JPSG

featute (attribute) decides whether D works as a complement or as a modifier. Note that each grammar, called a *structural principle*, is expressed as the constraints among three features, M, D, and H, in the local phrase structure tree.

As shown in the above definition, feature structures are very similar to the data structure in DOOD [12]. We will see some requirements of natural language processing for our DOOD system and develop applications on the DOOD system.

[11] See the details in [Tsuda 1992].

[12] This is one of the reason why we decided to design *Quixote*. See the appendix.

5.1.2 cu-Prolog

In order to process feature structures efficiently, we have developed a new CLP called *cu-Prolog*. A rule is defined as follows [13]:

$$H \Leftarrow B_1, \cdots, B_n || C_1, \cdots, C_m.$$

where $H, B_1, \cdots, B_n$ are atomic formulas, whose arguments can be in the form of feature structures and $C_1, \cdots, C_m$ are constraints in the form of an equation among feature structures, variables, and atoms, or an atomic formula defined by another set of rules. There is a restriction for an atomic formula in constraints in order to guarantee the congruence of constraint solving. This can be statically checked. The semantic domain is a set of relations of partially tagged trees, as in CIL[Mukai 1988] and the constraint domain is also the same.

The derivation in cu-Prolog is a sequence of a pair (G, C) of a set of subgoals and a set of constraints, just as in conventional CLP. Their differences are as follows:

- All arguments in predicates can be feature structures, that is, unification between feature structures is necessary.
- A computation rule does not select a rule which does not contribute to constraint solving: in the case of $(\{A\} \cup G, C)$, $A' \Leftarrow B||C'$, and $A\theta = A'\theta$, the rule is not selected if a new constraint $C\theta \cup C'\theta$ cannot be reduced.
- The constraint solver is based on unfold/fold transformation, which produces new predicates dynamically in a constraint part.

'Disjunction' in feature structures of cu-Prolog is treated basically as 'conjunction', just as in an o-term in $\mathcal{QUIXOTE}$ and a nested term in Kappa (CRL). However, due to the existence of a predicate, disjunction is resolved (or unnested) by introducing new constraints and facts:

$$H \Leftarrow p([l = \{a, b\}]) \iff H \Leftarrow p([l = X])||\{new\_p(X)\}.$$
$$new\_p(a).$$
$$new\_p(b).$$

That is, in cu-Prolog, disjunctive feature structures are processed in OR-parallel, in order to avoid set unification as in CRL. Only by focusing on the point does the efficiency seem to depend on whether we want to obtain all solutions or not.

One of the distinguished features in cu-Prolog is dynamic unfold/fold transformation during query processing, which contributes much to improving the efficiency of query processing. Some examples of a JPSG parser in cu-Prolog appear in [Tsuda 1992]. As predicate-based notation is not essential, language features in cu-Prolog can be encoded into the specification of $\mathcal{QUIXOTE}$ and the constraint solver can also be embedded into the implementation of $\mathcal{QUIXOTE}$ without changing semantics.

[13] As we are following with the syntax of $\mathcal{QUIXOTE}$, the following notation is different from cu-Prolog.

5.2 Dynamical Programming

This work aims to extend a framework of constraint throughout computer and cognitive sciences [14]. In some sense, the idea originates in the treatment of constraints in cu-Prolog. Here, we describe an outline of dynamical programming as a general framework of treating constraints and an example in natural language processing.

5.2.1 Dynamics of Symbol Systems

As already mentioned in Section 2, partial information plays an essential role in knowledge information processing systems. So, knowing how to deal with the partiality will be essential for future symbol systems. We employ a constraint system, which is independent of information flow. In order to make the system computationally more tractable than conventional logic, it postulates a *dynamics* of constraints, where the state of the system is captured in terms of *potential energy*.

Consider the following program in the form of clauses:

$$p(X) \Leftarrow r(X, Y), p(Y).$$
$$r(X, Y) \Leftarrow q(X).$$

Given a query $?\text{-}p(A), q(B)$, the rule-goal graph as used in deductive databases emulates top-down evaluation as in Figure 16. However, the graph presupposes a cer-

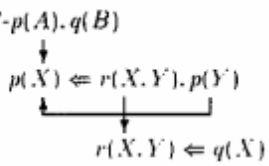


Figure 16: Rule-Goal Graph

tain information flow such as top-down or bottom-up evaluation. More generally, we consider it in the form in Figure 17, where the lines represent (partial) equations among variables, and differences between variables are not written for simplicity. We call such a graph a *constraint network*.

In this framework, computation proceeds by propagating constraints in a node (a variable or an atomic

[14] See the details in [Hasida 1992].

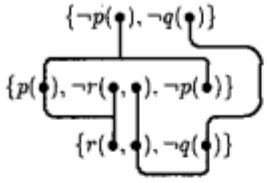


Figure 17: Constraint network

constraint) to others on the constraint network. In order to make such computation possible, we note the dynamics of constraints, as outlined below:

1) An *activation value* is assigned to each atomic constraint (an atomic formula or an equation). The value is a real number between 0 and 1 and is considered as the truth value of the constraint.

2) Based on activation values, *normalization energy* is defined for each atomic constraint, *deduction energy* and *abduction energy* are defined for each clause, and *assimilation energy* and *completion energy* are defined for possible unifications. The *potential energy* U is the sum of the above energies.

3) If the current state of a constraint is represented in terms of a point x of Euclidean space, U defines a *field of force* F of the point x. F causes *spreading activation* when $F \neq 0$. A change of x is propagated to neighboring parts of the constraint network, in order to reduce U. In the long run, the assignment of the activation values settles upon a stable equilibrium satisfying $F = 0$.

Symbolic computation is also controlled on the basis of the same dynamics. This computational framework is not restricted in the form of Horn clauses.

5.2.2 Integrated Architecture of Natural Language Processing

In traditional natural language processing, the system is typically a sequence of syntactic analysis, semantic analysis, pragmatic analysis, extralinguistic inference, generation planning, surface generation, and so on. However, syntactic analysis does not necessarily precede semantic and pragmatic comprehension, and generation planning is entwined with surface generation. Integrated architecture is expected to remedy such a fixed information flow. Our dynamics of constraint is appropriate for such an architecture.

Consider the following example:

Tom took a telescope. He saw a girl with it.

We assume that *he* and *it* are anaphoric with *Tom* and *the telescope*, respectively. However, *with it* has attachment ambiguity:

Tom has a telescope when he sees the girl. or *the girl has the telescope when Tom sees her.*

Consider a set of facts:

(1) $take(tom, telescope)$.

(2) $have(tom, telescope)$.

(3) $have(girl, telescope)$.

and an inference rule:

(4) $have(X, Y) \Leftarrow take(X, Y)$.

By constructing the constraint networks of (1),(2),(4) and (1),(3),(4) as in Figure 18, we can see that there

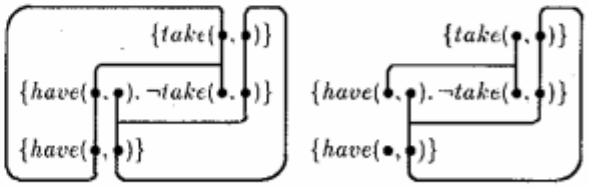


Figure 18: Constraint Networks of Alternatives

are two cycles (involving *tom* and *telescope*) in the left network ((1), (2), and (4)), while there is only one cycle (*girl*) in the right network ((1), (3), and (4)). From the viewpoint of potential energy, the former tends to excite more strongly than the latter, in other words, (2) is more plausible than (3).

Although, in natural language processing, resolution of ambiguity is a key point, the traditional architecture has not been promising, while our integrated architecture based on a dynamics of constraint network seems to give more possibilities not only for such applications but also for knowledge-base management systems.

6 Related Works

Our database and knowledge-base management system in the framework of DOOD has many distinguished features in concept, size, and varieties, in comparison with other systems. The system aims not only to propose a new paradigm but also to provide database and knowledge-base facilities in practice for many knowledge information processing systems.

There are many works, related to DOOD concepts, for embedding object-oriented concepts into logic programming. Although *F-logic*[Kifer and Lausen 1989] has the richest concepts, the *id-term* for object identity is based on predicate-based notation and properties are insufficient from a constraint point of view. Furthermore, it lacks update functions and a module concept.

Quixote has many more functions than F-logic. Although, in some sense, *Quixote* might be an over-specification language, users can select any subclass of *Quixote*. For example, if they use only a subclass of object terms, they can only be conscious of the sub-language as a simple extension of Prolog.

As for nested relational models, there are many works since the proposal in 1977, and several models have been implemented: *Verso* [Verso 1986], *DASDBS* [Schek and Weikum 1986], and *AIM-P* [Dadam *et al.* 1986]. However, the semantics of our model is different from theirs. As the (extended) NF^2 model of DASDBS and AIM-P has set-based (higher order) semantics, it is very difficult to extend the query capability *efficiently*, although the semantics is intuitively familiar to the user. On the other hand, as Verso is based on the universal relation schema assumption, it guarantees efficient procedural semantics. However, the semantics is intuitively unfamiliar to the user: even if $t \notin \sigma_1 T$ and $t \notin \sigma_2 T$ for a relation T, it might happen that $t \in \sigma_1 T \cup \sigma_2 T$. Compared with them, Kappa takes simple semantics, as mentioned in Section 3. This semantics is retained in o-terms in *Quixote* and disjunctive feature structures in cu-Prolog for efficient computation.

As for genetic information processing, researchers in logic programming and deductive databases have begun to focus on this area as a promising application. However, most of these works are devoted to query capabilities such as transitive closure and prototyping capabilities, while there are few works which focus on data and knowledge representation. On the other hand, *Quixote* aims at both the above targets. As for legal reasoning, there are many works based on logic programming and its extensions. Our work has not taken their functions into consideration, but has reconsidered them from a database point of view, especially by introducing a module concept.

7 Future Plans and Concluding Remarks

We have left off some functions due to a shortage in man power and implementation period. We are considering further extensions through the experiences of our activities, as mentioned in this paper.

First, as for *Quixote*, we are considering the following improvements and extensions:

- Query transformation techniques such as sideways information passing and partial evaluation are not fully applied in the current implementation. Such optimization techniques should be embedded in *Quixote*, although constraint logic programming needs different devices from conventional deductive databases. Furthermore, for more efficient query processing, flexible control mechanisms, such as in cu-Prolog and dynamical programming, would be embedded.

- For more convenience for description in *Quixote*, we consider meta-functions as HiLog [Chen *et al.* 1989]:

  $$tc(R)(X,Y) :- R(X,Y)$$
  $$tc(R)(X,Y) :- tc(R)(X,Z).tc(R)(Z,Y)$$

  In order to provide such a function, we must introduce new variables ranging over basic objects.

  This idea is further extended to a platform language of *Quixote*. For example, although we must decide the order relation (such as Hoare, Smyth, or Egli-Milner) among sets in order to introduce a set concept, the decision seems to depend on the applications. For more applications, such a relation would best be defined by a platform language. The current *Quixote* would be a member of a family defined in such a platform language.

- Communication among *Quixote* databases plays an important role not only for distributed knowledge-bases but also to support *persistent view*, *persistent hypothesis*, and *local* or *private* databases. Furthermore, cooperative query processing among agents defined *Quixote* is also considered, although it closely depends on the ontology of object identity.

- In the current implementation, *Quixote* objects can also be defined in KL1. As it is difficult to describe every phenomena in a single language, as you know, all languages should support interfaces to other languages. Thus, in *Quixote* too, a multi-language system would be expected.

- Although, in the framework of DOOD, we have focused mainly on data modeling extensions, the direction is not necessarily orthogonal from logical extensions and computational modeling extensions: set grouping can emulate negation as failure and the procedural semantics of *Quixote* can be defined under the framework of object-orientation.

  However, from the viewpoint of artificial intelligence, non-monotonic reasoning and 'fuzzy' logic should be further embedded, and, from the viewpoint of design engineering, other semantics such as object-orientation, should also be given.

As for Kappa, we are considering the following improvements and extensions:

- In comparison with other DBMSs by Wisconsin Benchmark, the performance of Kappa can be further improved, especially in extended relational

algebra, by reducing inter-kernel communication costs. This should be pursued separately from the objective.

- It is planned for Kappa to be accessed not only from sequential and parallel inference machines but also from general purpose machines or workstations. Furthermore, we should consider the portability of the system and the adaptability for an open system environment. One of the candidates is heterogenous distributed DBMSs based on a client-server model. although Kappa-P is already a kind of distributed DBMS.

- In order to provide Kappa with more applications, customizing facilities and service utilities should be strengthened as well as increasing compatibility with other DBMSs.

In order to make Kappa and *Quixote* into an integrated knowledge-base management system, further extensions are necessary:

- *Quixote* takes nested transaction logic, while Kappa takes flat transaction logic. As a result, *Quixote* guarantees persistence only at the top level transaction. In order to couple them more tightly, Kappa should support nested transaction logic.

- From the viewpoint of efficient processing, users cannot use Kappa directly through *Quixote*. This, however, causes difficulty with object identity, because Kappa does not have a concept of object identity. A mechanism to allow Kappa and *Quixote* to share the same object space should be considered.

- Although Kappa-P is a naturally parallel DBMS, current *Quixote* is not necessarily familiar with parallel processing, even though it is implemented in KL1 and works in parallel. For more efficient processing, we must investigate parallel processing in Kappa and *Quixote*.

We must develop bigger applications than those we mentioned in this paper. Furthermore, we must increase the comtatibility with the conventional systems: for example, from Prolog to *Quixote* and from the relational model to our nested relational model.

We proposed a framework for DOOD, and are engaged in various R&D activities for databases and knowledge-bases in the framework, as mentioned in this paper. Though each theme does not necessarily originate from the framework, our experiences indicate that this direction is promising for many applications.

Acknowledgments

The authors have had much cooperation from all members of the third research laboratory of ICOT for each topic. We especially wish to thank the following people for their help in the specified topics: Hiroshi Tsuda for *Quixote* and *cu-Prolog*, Moto Kawamura and Kazutomo Naganuma for *Kappa*, Hidetoshi Tanaka and Yuikihiro Abiru for *Biological Databases*, Nobuichiro Yamamoto for *TRIAL*. Satoshi Tojo for *Temporal Inference*, and Kôiti Hasida for *DP*.

We are grateful to members of the DOOD (DBPL, ETR, DDB&AI, NDB, IDB), STASS, and JPSG working groups for stimulating discussions and useful comments on our activities, and, not to mention, all members of the related projects (see the appendix) for their implementation efforts.

We would also like to acknowledge Kazuhiro Fuchi and Shunichi Uchida without whose encouragement *Quixote* and Kappa would not have been implemented.

References

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

[Chen *et al.* 1989] W. Chen, M. Kifer and D.S. Warren, "HiLog as a Platform for Database Language", *Proc. the Second Int. Workshop on Database Programming Language*, pp.121-135, Gleneden Beach, Oregon, June, 1989.

[Chikayama 1984] T. Chikayama, "Unique Features of ESP", *Proc. Int. Conf. on Fifth Generation Computer Systems*, ICOT, Tokyo, Nov.6-9, 1984.

[Chikayama *et al.* 1988] T. Chikayama, H. Sato, and T. Miyazaki,"Overview of the Parallel Inference Machine Operating Sistem (PIMOS)", *Proc. Int. Conf. on Fifth Generation Computer Systems*, ICOT, Tokyo, Nov.28-Dec.2, 1988.

[Dadam *et al.* 1986] P. Dadam, et al, "A DBMS Prototype to Support Extended NF^2 Relations: An Integrated View on Flat Tables and Hierarchies", *ACM SIGMOD Int. Conf. on Management of Data*, 1986.

[Delobel *et al.* 1991] 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.

[Goto *et al.* 1988] A. Goto *et al.*,"Overview of the Parallel Inference Machine Architecture (PIM)", *Proc.*

Int. Conf. on Fifth Generation Computer Systems, ICOT, Tokyo, Nov.28-Dec.2, 1988.

[Haniuda *et al.* 1991] H. Haniuda, Y. Abiru, and N. Miyazaki, "PHI: A Deductive Database System", *Proc. IEEE Pacific Rim Conf. on Communication, Computers, and Signal Processing*, May, 1991.

[Hasida 1992] K. Hasida, "Dynamics of Symbol Systems — An Integrated Architecture of Cognition". *Proc. Int. Conf. on Fifth Generation Computer Systems*, ICOT, Tokyo, June 1-5, 1992.

[Kawamura *et al.* 1992] M. Kawamura, H. Naganuma, H. Sato, and K. Yokota, "Parallel Database Management System *Kappa-P*", *Proc. Int. Conf. on Fifth Generation Computer Systems*, ICOT, Tokyo, June 1-5, 1992.

[Kifer and Lausen 1989] 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.

[Kim *et al.* 1990] 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.

[Miyazaki *et al.* 1989] N. Miyazaki, H. Haniuda, K. Yokota, and H. Itoh, "A Framework for Query Transformation", *Journal of Information Processing*, vol.12, No.4, 1989.

[Mukai 1988] K. Mukai, "Partially Specified Term in Logic Programming for Linguistic Analysis", *Proc. Int. Conf. on Fifth Generation Computer Systems*, ICOT, Tokyo, Nov.28-Dec.2, 1988.

[Schek and Weikum 1986] H.-J. Schek and G. Weikum, "DASDBS: Concepts and Architecture of a Database System for Advanced Applications", *Tech. Univ. of Darmstadt, Technical Report*, DVSI-1986-T1, 1986.

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

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

[Tsuda 1992] H. Tsuda, "cu-Prolog for Constraint-Based Grammar". *Proc. Int. Conf. on Fifth Generation Computer Systems*, ICOT, Tokyo, June 1-5, 1992.

[Ueda and Chikayama 1990]
K. Ueda and T. Chikayama, "Design of the Kernel Language for thr Parallel Ingerence Machine". *The Computer Journal*, vol.33, no.6, 1990.

[Verso 1986] J. Verso, "VERSO: A Data Base Machine Based on Non 1NF Relations". *INRIA Technical Report*, 523, 1986.

[Yamamoto 1990] N. Yamamoto, "TRIAL: a Legal Reasoning System (Extended Abstract)". *Joint French-Japanese Workshop on Logic Programming*, Renne, France, July, 1991.

[Yasukawa *et al.* 1992] 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.

[Yokota 1988] K. Yokota, "Deductive Approach for Nested Relations". *Programming of Future Generation Computers II*, eds. by K. Fuchi and L. Kott, North-Holland, 1988.

[Yokota *et al.* 1988] K. Yokota, M. Kawamura, and A. Kanaegami, "Overview of the Knowledge Base Management System (KAPPA)". *Proc. Int. Conf. on Fifth Generation Computer Systems*, ICOT, Tokyo, Nov.28-Dec.2, 1988.

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

[Yoshida 1991] K. Yoshida, "The Design Principle of the Human Chromosome 21 Mapping Knowledge-base (Version CSH91)". *Inetrnal Technical Report of Lawrence Berkley Laboratory*, May, 1991.

Appendix

Notes on Projects for Database and Knowledge-Base Management Systems

In this appendix, we describe an outline of projects on database and knowledge-base management systems in the FGCS project. A brief history is shown in Figure 19 [15]. Among these projects, Mitsubishi Electric Corp. has cooperated in Kappa-I, Kappa-II, Kappa-P, DO-ι, CIL, and *Quixote* projects, Oki Electric Industry Co., Ltd. has cooperated in PHI (DO-ϕ) and *Quixote* projects, and Hitachi, Ltd. has cooperated in ETA (DO-η) and *Quixote* projects.

a. Kappa Projects

In order to provide database facilities for knowledge information processing systems, a *Kappa* [16] project begun in September, 1985 (near the beginning of the intermediate stage of the FGCS project). The first target was to build a database with electronic dictionaries including concept taxonomy for natural language processing systems and a database for mathematical knowledge for a proof checking system called CAP-LA. The former database was particularly important: each dictionary has a few hundred thousands entries, each of which has a complex data structure. We considered that the normal relational model could not cope with such data and decided to adopt a nested relational model. Furthermore, we decided to add a new type *term* for handling mathematical knowledge. The DBMS had to be written in ESP and work on PSI machines and under the SIMPOS operating system. As we were afraid of whether the system in ESP would work efficiently or not, we decided on the semantics of a nested relation and started to develop a prototype system called *Kappa-I*. The system, consisting of 60 thousands lines in ESP, was completed in the spring of 1987 and was shown to work efficiently for a large amount of dictionary data. The project was completed in August, 1987 after necessary measurement of the processing performance.

After we obtained the prospect of efficient DBMS on PSI machines, we started the next project, *Kappa-II*[Yokota *et al.* 1988] in April, 1987, which aims at a practical DBMS based on the nested relational model. Besides the objective of more efficient performance than *Kappa-I*, several improvements were planned: a main memory database facility, extended relational algebra, user-definable command facility, and user-friendly window interface. The system, consisting of 180 thousand lines in ESP, works 10 times more efficiently in PSI-II machines than Kappa-I does in PSI-I. The project was over in March, 1989 and the system was widely released, not only for domestic organizations but also for foreign ones, and mainly for genetic information processing.

To handle larger amounts of data, a parallel DBMS project called *Kappa-P*[Kawamura *et al.* 1992] was started in February, 1989. The system is written in KL1 and works under an environment of PIM machines and the PIMOS operating system. As each local DBMS of Kappa-P works on a single processor with almost the same efficiency as Kappa-II, the system is expected to work on PIM more efficiently than *Kappa-II*, although their environments are different.

b. Deductive Database Projects

There were three projects for deductive databases.

First, in parallel with the development of Kappa, we started a deductive database project called *CRL* (complex record language) [Yokota 1988], which is a logic programming language newly designed for treating nested relations.

CRL is based on a subclass of complex objects constructed by set and tuple constructors and with a *module* concept. The project started in the summer of 1988 and the system, called *DO-ι*, was completed in November, 1989. The system works on *Kappa-II*. The query processing strategy is based on methods of generalized magic sets and semi-naive evaluation. In it, rule inheritance among modules based on submodule relations are dynamically evaluated.

Secondly, we started a project called *PHI* [Haniuda *et al.* 1991] in the beginning of the intermediate stage (April, 1985). This aimed at more efficient query processing in traditional deductive databases than other systems. The strategy is based on three kinds of query transformation called *Horn clause transformation (HCT)*[Miyazaki *et al.* 1989]: HCT/P executes partial evaluation or unfolding, HCT/S propagates binding information without rule transformation, and HCT/R transforms a set of rules in order to restrict the search space and adds related new rules. The HCT/R corresponds to the generalized magic set strategy. By combining these strategies, PHI aims at more efficient query processing. The consequent project is called *DO-ω*, in which we aim at a deductive mechanism for complex objects.

Thirdly, we started a project called *ETA* in April, 1988, which aimed at knowledge-base systems based on knowledge representation such as semantic networks. One year later, the project turned towards extensions of deductive databases and was called *DO-η*.

[15] At the initial stage of the FGCS project, there were other projects for databases and knowledge-based: *Delta* and *Kaiser*, however these were used for targets other than databases and knowledge-bases.

[16] A term *Kappa* stands for *knowledge application oriented advanced database management system*.

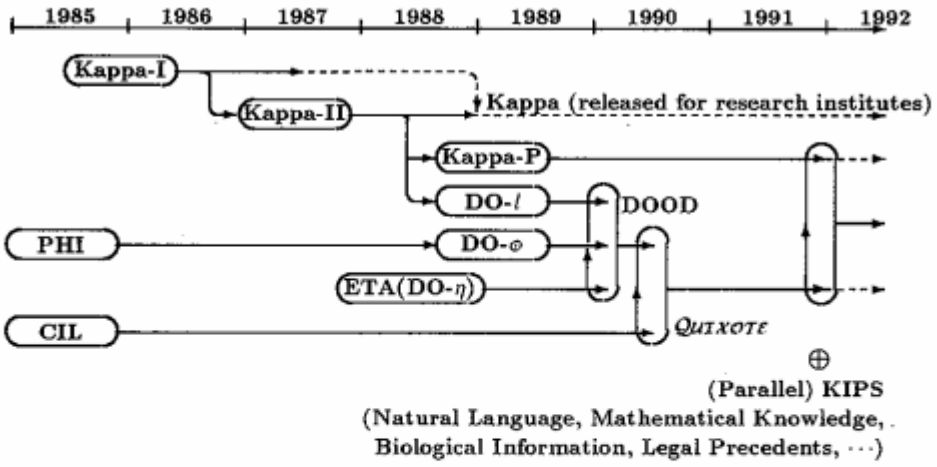


Figure 19: Brief History of Projects on Database and Knowledge-Base Management Systems

"*DO*" in the above projects stands for deductive and object-oriented databases and is shown to adopt a concept of DOODs [Yokota and Nishio 1989] as its common framework.

c. CIL Project

A language called *CIL* (complex indeterminates language) was proposed in April, 1985 [Mukai 1988]. The language aimed at semantic representation in natural language processing and was used not only in the discourse understanding system called *DUALS*, but also for representing various linguistic information. The implementation of CIL was improved several times and CIL was released to many researchers in natural language processing. The language is a kind of constraint logic programming and closely relates to situation theory and semantics. The language is based on *partially specified terms*, each of which is built by a tuple constructor. A set constructor was introduced into partially specified terms in another language *cu-Prolog*, as mentioned in Section 5.1.

d. *Quixote* Project

We tried to extend CRL not only for nested relations but also for DOODs, and to extend CIL for more efficient representation, such as the disjunctive feature structure. After these efforts, we proposed two new languages: *Juan*, as an extension of CRL, and *QUINT*, as an extension of CIL. While designing their specifications, we found many similarities between *Juan* and *QUINT*, and between concepts in databases and natural language processing, and decided to integrate these languages. The integrated language is *Quixote*[Yasukawa *et al.* 1992] (with Spanish pronunciation) [17]. As the result of integration, *Quixote* has various features, as mentioned in this paper. The *Quixote* project was started in August, 1990. The first version of *Quixote* was released to restricted users in December, 1991, and the second version was released for more applications at the end of March, 1992. Both versions are written in KL1 and work on parallel inference machines.

e. Working Groups on DOOD and STASS

At the end of 1987, we started to consider integration of logic and object-orientation concepts in the database area. After discussions with many researchers, we formed a working group for DOOD and started to prepare a new international conference on deductive and object-oriented databases [18]. The working group had four sub-working-groups in 1990: for database programming languages (DBPL), deductive databases and artificial intelligence (DDB&AI), extended term representation (ETR), and biological databases (BioDB). In 1991, the working group was divided into intelligent databases (IDB) and next generation databases (NDB). In their periodic meetings [19], we discussed not only problems of DOOD but also directions and problems

[17] Our naming convention follows the DON series, such as Don Juan and Don Quixote, where DON stands for "Deductive Object-Oriented Nucleus".

[18] Most of the preparation up until the first international conference (DOOD89) was continued by Professor S. Nishio of Osaka University.

[19] Their chairpersons are Yuzuru Tanaka of Hokkaido U. for DOOD, Katsumi Tanaka of Kobe U. for DBPL, Chiaki Sakama of ASTEM for DDB&AI and IDB, Shojiro Nishio of Osaka U. for ETR, Akihiko Konagaya of NEC for BioDB, and Masatoshi Yoshikawa of Kyoto Sangyo U. for NDB.

of next generation databases. These discussions contributed greatly to our DOOD system.

From another point of view, we formed a working group (STS) [20] for situation theory and situation semantics in 1990. This also contributed to strengthening other aspects of *Quixote* and its applications.

[20]The chairperson is Hozumi Tanaka of Tokyo Institute of Technology.