From Databases to Knowledge-Bases — *Kappa* , *QUIXOTE* , *Helios*

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

Abstract

In the FGCS project and its Follow-on project, we have designed and developed a nested relational database management system, *Kappa*, a deductive object-oriented database (DOOD) language (or a knowledge representation language), *QUIXOTE*, and a heterogeneous distributed cooperative problem solving system, *Helios*, for knowledge information processing applications. In this paper, from the viewpoint of database and knowledge-base management systems, I overview their objectives and features, and summarize their contributions. Especially. I focus mainly on their language aspects rather than systems. Further, by reflecting their experiences, I discuss some directions of future database from an application point of view.

1 Introduction

In the FGCS (Fifth Generation Computer Systems, 1982-1993) project[Kurozumi 92] and its Follow-on project (1993-1995)[Uchida *et al* 93], we have been engaged in various knowledge information processing applications such as natural language processing, genetic information processing, and legal reasoning. In such an environment, our major research interest from database and knowledge-base points of view has been how to support such applications by domain-independent languages and systems as database and knowledge-base management systems[Yokota and Yasukawa 92].

For the objective, we have two activities: how to represent data and knowledge effectively and manage them efficiently; how to provide an environment where multiple languages and systems can work effectively for single purpose. During two projects, we have designed and developed three database and knowledge-base management systems:

1. Nested relational database management system, *Kappa*[Yokota *et al* 88, Kawamura *et al* 92, Kawamura and Kawamura 94],
2. Deductive object-oriented database (DOOD) language/system, *QUIXOTE* [Yokota *et al* 93, Yokota *et al* 94, Yasukawa *et al* 92], and
3. Heterogeneous, distributed, cooperative problem solving system, *Helios*[Yokota and Aiba 93, Yokota 94, Aiba *et al* 94a].

The decision of the adoption of a nested relational data model in *Kappa* is based on assessment of database technologies in the middle of 1980s for efficiently processing a large amount of structured data. *QUIXOTE*, which is based on a DOOD paradigm newly proposed in 1988, is not only an extension of deductive databases such as CRL[Yokota 88] and PHI[Haniuda *et al* 91], but also an extension of knowledge representation languages such as CIL[Mukai 88]. *Helios*'s approach is rather practical: most of its motivations comes from constructing large-scaled applications, where we must consider various heterogeneiy such as model heterogeneity, spatial heterogeneity, and temporal heterogeneity, and cooperation for resource bound environments.

From a database point of view, *Kappa* is a database engine which guarantees to process algebraic operations for various structured data efficiently, *QUIXOTE* is an upper layer of *Kappa* and make representation and management of higher-level data and knowledge such as rules possible, and *Helios* can be considered as an extension of multidatabase[Yokota 94], where multiple heterogeneous database management systems can process a query cooperatively.

From a knowledge information processing point of view, *Kappa* is a back-end system hidden from users, *QUIXOTE* provides a flexible knowledge representation language with both logic and object-orientation features and a thought-experimental environment in query processing for partial information, and *Helios* makes it possible to combine various representation languages and constraint solvers as a single problem solver.

We have taken a bottom-up approach for their development as *from databases to knowledge-bases*: that is, from *Kappa* to *QUIXOTE* and from *QUIXOTE* to *Helios*. According as their developments, we have extended applications incrementally to be larger-scaled and to satisfy more requirements.

The objective of this paper is not only to give each outline but also to discuss future generation database and knowledge-base systems by reflecting their experiences and related applications along the above approach. In sections 2,3,4, I describe outlines of *Kappa*, *Quixote*, and *Helios*, respectively, from their designer's point of view. In Section 6, I re-visit their design decisions from large-scaled knowledge information applications' point of view and discuss about future database and knowledge-base systems.

2 *Kappa* and *CRL*

2.1 Motivations

From a logic programming point of view in the middle of 1980s, the relational data model was most appropriate for the underlying database engine, because a predicate without functions just corresponds to a tuple in the relational model. However, when we aim to process efficiently a large amount of structured data such as natural language dictionaries and, later, genetic information databases, we had to abandon the relational model to avoid computational complexity such as join operations. In 1985, we adopted a nested relational model as the underlying database engine, because we had seen through its efficient implementations, and started up the *Kappa* project

The advantages of nested relational models over the relational model are that they offer more efficient representation and processing for structured data. Since the advantages were pointed out in [Makinouchi 77], there have been many works[Verso 86, Dadam *et al* 86, Schk and Weikum 86, Scholl and Schk 87, Deshpande and Gucht 88]. It is also widely known that nested relational models are better than the relational model for new applications such as engineering, office and geographical databases; and many commercial DBMSs also employ the idea from a practical point of view. There are two problems for nested relational model:

- As there are some variants of the name, the formal semantics should be made clear.
- As users must be conscious of nested structure, the semantics and operations should be considered to reduce users' burdens.

Our nested relational database management system is called *Kappa*, which has several implementations [Yokota *et al* 88, Kawamura *et al* 89, Kawamura *et al* 92, Kawamura and Kawamura 94].

A deductive database is an extension of the relational model by the proof-theoretic reconstruction of a relational database [Gallaire *et al* 84]. deductive databases can be considered as a first step towards knowledge bases.

By adoption of a nested relational model, we had to abandon ordinary logic programming languages and to design and develop a new logic programming language, *CRL*[Yokota 88, Kiyama and Yokota 88, Takahashi and Yokota 90].

In this section, I describe the outline of *Kappa* and its related *CRL*[Yokota 88].

2.2 Nested Relation in *Kappa*

Assume a set O of atomic objects, a set T of types of atomic objects, a set A of attribute names, a tuple constructor $[,\ ]$, and a set constructor $\{,\ \}$. O may contain a special object ω to represent explicitly a void or null object, which is used for partial information of a tuple object. An *object* is defined as follows:

Definition 1 Object

1. Any atomic object is an object.
2. If $o_1, \cdots, o_n$ are objects with a type τ, then $\{o_1, \cdots, o_n\}$ is an object of a type $\{\tau\}$, which is called a *set object*.
3. If $a_1, \cdots, a_n$ are different attribute names, $o_1, \cdots, o_n$ are objects with types $\tau_1, \cdots, \tau_n$, and any of $a_1, \cdots, a_n$ is not used in $o_1, \cdots, o_n$, then $[a_1 = o_1, \cdots, a_n = o_n]$ is an object of a type $[a_1 = \tau, \cdots, a_n = \tau_n]$, which is called a *tuple object*.

□

A tuple object is called a *nested tuple*, if some attribute value is not an atomic value. A *nested relation* is a set object (with a type $\{\tau\}$) consisting of nested tuples, where the *schema* is defined as a type $\{\tau\}$. And a *nested relational database* is a tuple object consisting of pairs of a relation name and a nested relation. It is easy to see that these definitions correspond with those of ordinary nested relations intuitively.

In this formulation, we can consider some kinds of semantics about a set constructor, especially connected with row-nest and row-unnest operations. To make the semantics clear, consider the following examples:

Example 1 Set Grouping
Assume there are two tuples: $[a = c_1, b = \{c_2, c_3\}]$ and $[a = c_1, b = \{c_3, c_4\}]$. There are two possible tuples after application of a row-nest operation to the given tuples:

$$[a = c_1, b = \{c_2, c_3, c_4\}], \quad \text{or}$$
$$[a = c_1, b = \{\{c_2, c_3\}, \{c_3, c_4\}\}].$$

Each tuple is resulted by a set union or a set-of (set grouping) operations respectively. □

While an (extended) NF^2 model[Dadam *et al* 86, Schk and Weikum 86] employs the second semantics, Verso model[Verso 86] employs the first. LDL [Zaniolo 88] is also considered as the second case. We take the first, and give the semantics of a nested tuple as a set of only column-nested tuples, which is independent of row-nest and row-unnest operation. Under the semantics, we can omit { and } in the case of a singleton set.

There is the following difference between our model and Verso:

Example 2 Consider a relation R:

$$\{[a = k_1, b = \{d_1, d_2\}, c = \{d_3, d_4\}]\}.$$

By selections $\sigma_{c=d_3}(R)$ and $\sigma_{b=d_1}(R)$, we can get the following relations respectively:

$$\begin{aligned} R_1 &= \{[a = k_1, b = \{d_1, d_2\}, c = d_3]\} \\ R_2 &= \{[a = k_1, b = d_1, c = \{d_3, d_4\}]\} \end{aligned}$$

By a union operation $R_1 \cup R_2$, Verso returns the original relation R_1, while our model returns the following:

$$\begin{aligned} &\{[a{=}k_1, b{=}d_1, c{=}\{d_3, d_4\}], [a{=}k_1, b{=}d_2, c{=}d_3]\}, \text{ or} \\ &\{[a{=}k_1, b{=}\{d_1, d_2\}, c{=}d_3], [a{=}k_1, b{=}d_1, c{=}d_4]\} \end{aligned}$$

according to the nested sequence. □

Verso guarantees more efficient query processing than *Kappa*, however Verso is more difficult to understand operations' result than *Kappa*.

Under the semantics, users do not need to be conscious of the row-nest structure when they query a database.

Example 3 Semantics of Nested Relation
According to the above semantics, all of the following relations have the same meaning:

$$\begin{aligned} &\text{R}_1\text{:} \quad \{[a = c_1, b = c_3], [a = c_2, b = c_3], [a = c_2, b = c_4]\}, \\ &\text{R}_2\text{:} \quad \{[a = \{c_1, c_2\}, b = c_3], [a = c_2, b = c_4]\}, \quad \text{and} \\ &\text{R}_3\text{:} \quad \{[a = c_1, b = c_3], [a = c_2, b = \{c_3, c_4\}]\} \end{aligned}$$

$[a = \{c_1, c_2\}, b = c_3]$ is implied by each of them. □

A nested relational model based on such semantics is a natural extension of a relational model, and its characteristics are more efficient representation and more efficient processing performance. The extended relational algebra including nest and unnest operations is reconstructed according to the semantics.

Strictly speaking, such relations may be classified into *unnormalized* relations and *nested* relations from a structural point of view, depending on whether the relations can be reversible by row-unnest and row-nest operations, and also classified into *value-oriented* relations and *expression-oriented* relations from an operational point of view, depending on whether sets have intrinsic meaning or not. Although in this context we do not discriminate the differences explicitly, *Kappa* treats *unnormalized* and *expression-oriented* relations in the above classification.

The actual model of *Kappa* has some additional features for practical use: list and bag constructors, term as an atomic object and its retrieval by unification or pattern matching, and use of a constraint logic programming language CAL [Aiba *et al* 88] as generation or integrity rules for algebraic constraints. The extended relational algebra corresponds to such extended features, and furthermore supports some convenient operators, besides conventional operations, for efficient processing.

2.3 *CRL* as a Bridge to Knowledge-Bases

CRL uses an attribute-valued notation instead of a predicate notation as in *Kappa*. Assume a set of variables besides the symbols of the nested tuples, the *term* is defined as follows:

Definition 2 CRL Term

1. An atomic object o is a term,
2. A variable X is a term,
3. If $t_1, \cdots, t_n$ are terms, then $\{t_1, \cdots, t_n\}$ is a term, which is called a *set term*,
4. If $a_1, \cdots, a_n$ are attribute names and $t_1, \cdots, t_n$ are terms, then $[a_1 = t_1, \cdots, a_n = t_n]$ is a term, which is called a *tuple term*.

□

Although *CRL* is a typeless language, the same restrictions as nested tuples are also imposed on the terms for nested relations. In the sense, *CRL* is less expressive than other languages for complex objects such as COL[Abiteboul and Grumbach 88].

This attribute-valued notation is an extension of a usual predicate notation by introducing a set of some attribute names:

$$p(t_1, \cdots, t_n) \Rightarrow [\$0 = p, \$1 = t_1, \cdots, \$n = t_n],$$

where $\$0, \$1, \cdots, \$n$ are newly introduced attribute names. The advantages are flexibility in the position and the number of arguments, and natural representation of column and row nesting tuples by set and tuple constructors.

The semantics of the term is defined as a set of *partially tagged trees* (*PTT*s), which is used in the semantics of CIL [Mukai 88]. The semantics reserves the semantics of nested tuples. According to the semantics, row-nest and row-unnest operations correspond to a distributive law under an algebraic structure with tuple and set constructors:

$$\{[a=c_1, b=\{c_2, c_3\}]\} \Leftrightarrow \{[a=c_1, b=c_2], [a=c_1, b=c_3]\}.$$

A *program clause* is defined as a pair of a tuple term t and a set $\{t_1, \cdots, t_n\}$ of tuple terms as follows:

$$t \leftarrow t_1, \cdots, t_n.$$

A *CRL program* is a set of program clauses. And a *goal* is a set of tuple terms $q_1, \cdots, q_m$, which is written as $\leftarrow q_1, \cdots, q_m$.

Example 4 CRL Program

$$\begin{array}{l} [par = \{\text{"}mary\text{"}\}, chi = \{\text{"}john\text{"}, \text{"}lisa\text{"}\}], \\ [par = \{\text{"}paul\text{"}, \text{"}kate\text{"}\}, chi = \{\text{"}mary\text{"}\}], \\ [anc = X, des = Y] \leftarrow [par = X, chi = Y], \\ [anc = X, des = Y] \leftarrow [par = X, chi = Z], \\ \qquad\qquad\qquad\qquad\quad [anc = Z, des = Y] \end{array}$$

is a CRL program. □

A *CRL database* is a CRL program and is divided into an intensional and an extensional databases (an IDB and an EDB), just like a definite clause based deductive database. An EDB is a set of nested tuples, which can correspond to a *Kappa* database.

The semantics shows the following properties:

$$\begin{array}{l} q \leftarrow p_1, \cdots, p_n \\ \equiv q \leftarrow p_1, \cdots, p_{i-1}, p_{i1}, \cdots, p_{im}, p_{i+1}, \cdots, p_n \\ \equiv q_1 \leftarrow p_1, \cdots, p_n \wedge \cdots \wedge q_k \leftarrow p_1, \cdots, p_n, \end{array}$$

where $p_{i1}, \cdots, p_{im}$ and $q_1, \cdots, q_k$ are 'unnested' results of p_i and q, respectively. The relation guarantees to reserve the declarative and procedural semantics of Prolog. As the extended SLD resolution for the CRL program, set unification (intersection) is applied for an EDB term without being unnesting, and the new goal corresponding to the set difference is generated for an IDB. For example, consider the following goal:

$$\leftarrow G_1, [\cdots, a = S, \cdots], G_2,$$

where S is a set term. If the subgoal $[\cdots, a = S, \cdots]$ can be unified with an EDB term $[\cdots, a = S', \cdots]$ by a unifier θ, the new goal is generated as follows:

$$\leftarrow (G_1, [\cdots, a = (S \setminus S'), \cdots], G_2)\theta,$$

if $S \setminus S'$, called *a residue goal*, is not an empty set.

As for bottom-up evaluation, the least fixpoint semantics like Prolog is also defined, and similar optimization strategies can be applied.

Example 5 Query Transformation
For a query $\leftarrow [anc = \{\text{"}paul\text{"}, \text{"}kate\text{"}\}, des = X]$ $(\leftarrow [anc = \text{"}paul\text{"}, des = X], [anc = \text{"}kate\text{"}, des = X])$, the above CRL database (program) can be transformed into the following form:

$$\begin{array}{l} [par = \{\text{"}mary\text{"}\}, chi = \{\text{"}john\text{"}, \text{"}lisa\text{"}\}]. \\ [par = \{\text{"}paul\text{"}, \text{"}kate\text{"}\}, chi = \{\text{"}mary\text{"}\}]. \\ [anc = X, des = Y] \leftarrow [anc^* = X], [par = X, chi = Y]. \\ [anc = X, des = Y] \leftarrow [anc^* = X], [par = X, chi = Z], \\ \qquad\qquad\qquad\qquad\qquad\qquad [anc = Z, des = Y]. \\ [anc^* = \text{"}paul\text{"}]. \\ [anc^* = \text{"}kate\text{"}]. \\ [anc^* = Z] \leftarrow [anc^* = X], [par = X, chi = Z]. \end{array}$$

□

This is an example of HCT/R [Miyazaki *et al* 89] for a CRL database.

A modular concept with rule inheritance is introduced to classify a set of rules with alternative definitions and inconsistency [Takahashi and Yokota 90]. Further query optimization is investigated [Kiyama and Yokota 88].

3 *QUIXOTE*

3.1 Motivations

During the development of *CRL* databases, we realized requirements of more powerful and flexible databases for *real* data-intensive applications. Another kind of requirements are how to integrate *CRL* and *CIL* for natural language processing. The keywords are *partial information*, *object-orientation*, and *constraints*. For the objective, we proposed a new paradigm, deductive object-oriented database (DOOD) [Yokota and Nishio 89, Yokota and Nishio 90].

From the viewpoint of knowledge representation, the above paradigm corresponds not only to value-based representation but also to identity-based representation in the sense of [Ullman 87]. As the identity of partial information is naturally treated in the context of object-orientation, we intended to integrate logic and object-orientation concepts in the context of constraint logic programming.

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

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

After many adventures of several prototype languages, we named a new language *QUIXOTE*, where we expect many *real* applications to be naturally modeled. Servantes wrote:

> *..., for a knight-errant who was loveless was a tree without leaves and fruit — a body without soul. ([Starkie 57], p.18)*

QUIXOTE would be read in this spirit.

3.2 Policies

In order to integrate various features into *QUIXOTE*, we consider various criteria. Here we summarize several points:

1. *Are all properties of an object homogeneous?*

This criterion concerns the classification of properties. For example, two phrases, "$\cdots$ *apple which is red* $\cdots$" and "$\cdots$ *apple, which is red,* $\cdots$" are different, that is, although *which is red* is common, the former is restricted reading, that is, *intrinsic*, while the latter is non-restricted reading, that is, incidental or *extrinsic*, where *apple* plays the role of identity.

2. *How is an object identity represented?*

This criterion concerns the representation of an object identity, i.e., object identifier (oid). Traditionally, an oid is represented in the form of a pointer or an address and is hidden from users in most object-oriented languages, while a name, corresponding to an oid, is explicitly used by users. We take a term consisting of intrinsic properties as a name, which also can be an oid for users, as in [Kifer and Lausen 89].

3. *How is extrinsic properties represented?*

This criterion concerns representation of extrinsic properties and the relation between oids. The representation of partial information should be flexible. Given an oid o, a label l, and a value v, an extrinsic property is represented as a constraint between $o.l$ (l-value of o) and v. We take the *subsumption* (a kind of ISA) relation as the relation, consider a constraint solver over a set of subsumption constraints, and take the idea in DOT [Tsukamoto *et al* 91] as the inheritance mechanism.

4. *Which semantics should be selected for sets and how are they treated?*

This criterion concerns how to treat sets, that is, ordering of sets, set grouping or set construction, and representation of complex objects. We selected Hoare ordering, because it seems to be natural in most of our applications when considering the subsumption relation. Set constructors are introduced into subsumption constraints and make it possible to represent complex objects. Sets in intrinsic properties are made to corresponds to the semantics of *Kappa* and *CRL*.

5. *How should globality and locality of data and knowledge be introduced?*

This criterion concerns the globality and locality of data and knowledge in a database, which also relate to the construction of very large knowledge-bases. To realize the coexistence of inconsistent knowledge or localization of properties or methods, we introduce a *module* concept as in [Takahashi and Yokota 90, Miller 86, Monterio and Porto 89, Monterio and Porto 90].

6. *What extensions are needed for query processing?*

This criterion concerns extensions of query processing for partial information. Focusing on the partiality, we introduce (restricted) abduction (or hypothesis generation) and hypothetical reasoning (conditional query processing). Such features provide databases with a thought-experiment environment.

7. *What about relationship to traditional object-orientation concepts?*

Most of object-orientation concepts such as object identity, property inheritance, and method can be supported by subsumption constraints. Further we can take a class by ordering between terms, however postpone to introduce encapsulation, a type system, and autonomy.

8. *What about relationship to deductive databases?*

Although negation and disjunction have been well studied in logic programming and deductive databases, subsumption constraints with negation or disjunction have not been studied. Thus, we introduce only negation-as-failure of terms and disequations of constraints. To retain upwards compatibility with deductive databases, we decided also to employ ordinary predicate notations as terms in *QUIXOTE*.

9. *What about relationship to database management features?*

We introduce update, nested transactions, persistence, and integrity constraints, however introduce only minimal features of concurrency control.

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

We discussed whether *QUIXOTE* might be quixotic or not (although the name was derived differently) during the design, because the language must have many features. We consider this integration to be rather moderate but not an unreasonable integration. As a result of considerations on many applications, *Quixote* has many aspects: a DOOD language, a knowledge representation language, a constraint logic programming language, a database programming language, and a situated programming language.

3.3 Basic Features of *Quixote* Objects

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

3.3.1 Object Identity and Subsumption Constraints

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

$$\begin{aligned}&apple,\\&apple[color = red], \text{ and}\\&cider[alcohol = yes,\\&\quad product = process[source = apple,\\&\qquad\qquad process = ferment]]\end{aligned}$$

are object terms, where *apple* is *basic*, but the latter two are *complex*. Although an infinite structure and set constructors are introduced, as explained in the second and fourth criteria in Section 3.2, we do not explain them here for simplicity.

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

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

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

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

$$\begin{aligned}apple|\{&apple.species \sqsubseteq rose,\\&apple.area \sqsupseteq_H \{aomori, nagano\}\},\end{aligned}$$

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

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

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

3.3.2 Property Inheritance

For *property inheritance*, we assume the following rule as in [Tsukamoto *et al* 91]:

> if $o_1 \sqsubseteq o_2$, and neither o_1 nor o_2 has labels, l and l'
> then $o_1.l \sqsubseteq o_2.l$ and $o_1.l' \sqsubseteq_H o_2.l'$,

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

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

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

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

Intrinsic properties interrupt property inheritance as follows:

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

This corresponds to *exception* of property inheritance.

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

$$\begin{array}{lcl}p.l \sqsubseteq a \wedge o.l \sqsubseteq b & \Rightarrow & o.l \sqsubseteq a \downarrow b\\ a \sqsubseteq o.l \wedge b \sqsubseteq o.l & \Rightarrow & a \uparrow b \sqsubseteq o.l\\ o.l \sqsubseteq_H s_1 \wedge o.l \sqsubseteq_H s_2 & \Rightarrow & o.l \sqsubseteq_H s_1 \cup s_2\\ s_1 \sqsubseteq_H o.l \wedge s_2 \sqsubseteq_H o.l & \Rightarrow & \{x \downarrow y \mid x \in s_1, y \in s_2\} \sqsubseteq_H o.l\end{array}$$

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

3.3.3 Intensional Objects

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

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

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

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

$$o|\{o.l \sqsubseteq a\} \Leftarrow \text{ and } o|\{o.l \sqsubseteq b\} \Leftarrow,$$

can be merged as follows:

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

In cases where an object term with variables is in the head of a rule, an object is defined when the variables are instantiated during query processing, because an object term with variables cannot specify an object. That is, subsumption constraints in a head are merged after evaluation of all the related rules.

When the constraints of a head are empty, *QUIXOTE* is an instance of CLP(X) [Jaffar and Lassez 87]:

$$\begin{aligned} o_0 \Leftarrow o_1|C_1, \cdots, o_n|C_n \parallel C \\ \iff o_0 \Leftarrow o_1, \cdots, o_n \parallel C_1 \cup \cdots \cup C_n \cup C \end{aligned}$$

From a programming language point of view, the existence of head constraints in a rule makes *QUIXOTE* an extension of CLP(X) and, in the procedural semantics, the possibility of merging head constraints must be checked at every OR node (in the sense of ordinary logic programming languages) of the resolution tree. See the details in [Yasukawa and Yokota 90].

3.3.4 Modules and Databases

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

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

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

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

The definitions mean that m_1 inherits all rules in m_2 and m_3, and m_2 inherits all rules in m_4. We call such inheritance *rule inheritance*, where exception, locality, and overriding are also defined [Yasukawa *et al* 92]. The submodule relation is an acyclic directed graph, in which modules can be nested.

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

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

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

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

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

Although the module is introduced as the fifth criterion in Section 3.2, we can list three objectives:

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

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

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

3.3.5 Query Processing

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

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

Further, as derivation becomes more complicated, we want to know why a particular answer is returned. As details of such query processing are presented in [Yokota *et al* 94], we outline the processing here.

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

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

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

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

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

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

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

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

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

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

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

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

3.3.6 Other Features

Here, we list some more features of *Quixote*:

- *Assertion* and *deletion* of extensional objects and properties during query processing are supported as in [Zaniolo 88]. These are controlled by the same uniform nested transaction logic used in hypothetical reasoning. However, note that the subsumption relation and submodule relation may not be updated during query processing, because the change in their inheritance might destroy the soundness of the derivation, although they may be statically inserted as hypotheses.
- All objects in *Quixote* except temporarily created objects during query processing are persistent. Persistent objects in a database are stored through a uniform logical interface into other database management systems or file systems via TCP/IP protocol. Such objects are invoked when the corresponding database is opened.
- A *Quixote* system consists of a client as a user interface and a server as a knowledge-base engine. Servers and clients are connected also by TCP/IP protocol and servers control multi-user access.

4 Helios

4.1 Motivations

As mentioned in Section 1, it is very difficult to cope with *real* large-scaled applications in a single language and a single paradigm. For example, although the current *Quixote* has many powerful features, it is very inefficient to represent various kinds of data and knowledge such as sequence data, algebraic constraints, and statistical data in it.

For example, consider a very simple one "a person who is over twenty years old can drink domestic alcoholic drink", which is written in $\mathcal{QUIXOTE}$ as follows:

$$\begin{aligned} person[name = X]/[canDrink = Y] \Leftarrow\ & Y/[product\_in = domestic], \\ & Y \sqsubseteq alcohol, \\ & person[name = X].age \geq 20 \end{aligned}$$

where "$Y \sqsubseteq alcohol$" is a subsumption constraint, while "$person[name = X].age \geq 20$" is an algebraic constraint, which can be processed, for example, in GDCC[Aiba and Hasegawa 92]. The problem is how $\mathcal{QUIXOTE}$ sends the algebraic constraint to GDCC and gets its answer from GDCC.

As another example, consider a genome database. Some knowledge such as protein functions are written in the form of rules in $\mathcal{QUIXOTE}$ and large volume of structures data, which can be stored in the form of nested relations in *Kappa*, are also represented in $\mathcal{QUIXOTE}$. However large-sized sequence specified as a value must be processed by information retrieval functions such as string search, which are supported by native functions of neither *Kappa* and $\mathcal{QUIXOTE}$. The problem is how to process a query in $\mathcal{QUIXOTE}$, *Kappa*, and information retrieval functions cooperatively.

As shown in the above examples, we can find many applications which require multiple *heterogeneous* problem solvers [1]:

- Modeling Heterogeneity
  The complexity of a given problem requires a combination of multiple heterogeneous problem solvers,
- Spatial Heterogeneity:
  Spatially distributed problem solvers are required to process a given problem.
- Temporal Heterogeneity:
  For a new problem, a new problem solver is not necessarily developed: that is, multiple existing problem solvers must be reused for a new problem.

There have been some approaches: an arithmetic calculator in Prolog and a constraint logic programming language with a single constraint solver. Such a restricted approach seems to be neither flexible nor promising for most applications.

Further, considering the spread of distributed environments, there might be similar resources, each of which does not have complete information. In such an environment, we can frequently get better results by accessing and merging multiple information resources or multiple problem solvers. In other words, cooperation among distributed resources are frequently required. Considering such applications and environments, heterogeneous, distributed, cooperative problem solvers will become more important and can play a role of a very large knowledge-base.

[1]Here, we use a *problem solver* as a general term for a database system, a knowledge-base system, a constraint solver, an expert system, an application program, and so on.

4.2 *Helios* Model

A basic concept in *Helios* is an *agent*, defined as follows:

$$\begin{aligned} \text{agent} \ := \ & (\text{capsule, problem-solver}) \\ | \ & (\text{capsule, environment,} \\ & \ \{\text{agent}_1, \cdots, \text{agent}_n\}) \end{aligned}$$

A *simple* agent is defined as a pair of a *capsule* and a problem solver: intuitively, a problem solver is wrapped with a capsule as in Figure 1. A *complex* agent is defined as a triple of a capsule, an *environment*, and a set of agents (agent$_1$,$\cdots$,agent$_n$), where an environment is a field where agent$_1$,$\cdots$,agent$_n$ can exist and communicate with each other. Intuitively, as a pair of an environment and a set of agents can be considered also as a problem solver, a new agent can be defined by wrapping them by a capsule. That is, an agent can be also hierarchically organized. Figure 1 shows such structures.

A capsule and an environment is defined as follows:

$$\begin{aligned} \text{capsule} \ := \ & (\text{agent-name,methods,self-model,} \\ & \ \text{negotiation-strategy}) \\ \text{environment} \ := \ & (\text{agent-names,common-type-system,} \\ & \ \text{negotiation-protocol,ontology}) \end{aligned}$$

An agent name in a capsule is an identifier of the corresponding agent and *agent names* in an environment specifies what agents exist in the environment. *Methods* in a capsule define *import* and *export* method protocols of the corresponding agent. An agent with only import methods is called *passive* and an agent with both methods is called *active*: that is, only an agent which send new messages through export methods can negotiate with other agents. A *common type system* in an environment enforces all agents under the environment to type all messages strongly. A *self model* in a capsule defines what the agent can do. An environment extracts necessary information from self models in agents to dispatch messages among agents. Under a *negotiation protocol* in an environment, each agent defines a *negotiation strategy* to communicate with other agents. An *ontology* defines the transformation of the contents of messages among agents, while a capsule convert the syntax and type of messages between the common type system and the intrinsic type system of the corresponding problem solver. These information is defined in *CAPL* (CAPsule Language) and *ENVL* (ENVironment Language).

Although various information is defined in each environment and each agent, a *message* among agents is in the form of a global *communication protocol* consisting of the message identifier, the identifier of a sender agent, the identifier of a receiver agent, a transaction identifier, and a message. A message identifier is

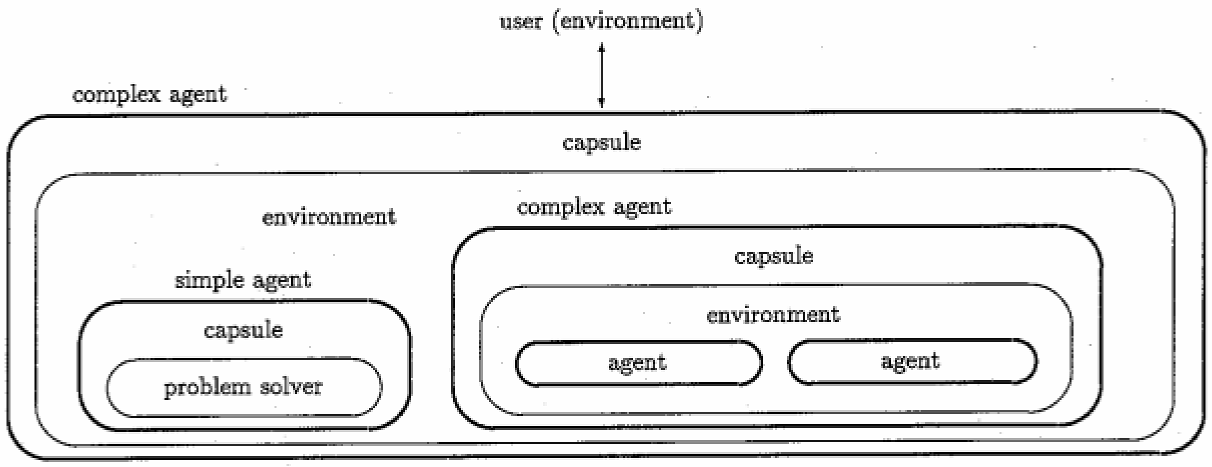

Figure 1: Basic Model of ***Helios***

common in a query message and answer messages. A transaction identifier is used to identify a negotiation process as a transaction, which can be nested.

4.3 Features of *Helios*

In this subsection, I explain several specific features of *Helios*, some of which have not been developed yet.

4.3.1 Message Dispatching

Any active agent can send a message to its environment. How is the message dispatched by the environment?

First, during the initialization of agent processes in the environment, the environment constructs a map of a logical agent name and a physical process address (or IP address). Secondly, the environment gathers method information and function information in self models from each agent and constructs two kinds of maps: a method and an agent name; a function name and an agent name. Such maps work for dispatching messages among agents.

As a method or a function does not necessarily corresponds to an agent uniquely, a message is possibly sent to multiple agents. This mechanism is useful for the followings:

- It is unnecessary to specify an agent name in problem solvers explicitly.
- It is possible to send simultaneously a message to possible agents.

An environment decides to send a message sequentially or in parallel to candidates listed by the maps, and processes answers sequentially or by grouping as a set. In the case of set grouping, aggregation functions can be specified in an environment. Such a mode can be selected in a query message.

4.3.2 Negotiation Protocol

How can agents communicate with each other? We consider three kinds of modes: *simple communication*, *negotiation-based communication*, and *schedulable communication*.

When communication among agents requires neither negotiation nor query plans, it is called simple.

For negotiation among agents, negotiation protocol and negotiation strategy in *Helios* can be defined in ENVL and CAPL, respectively, differently from conventional systems [Aiba *et al* 94b]. The protocol is based on the *transaction-based protocol*, which comes from the similarity between transaction and negotiation processes: negotiations can be nested and controlled by begin-, end-, and abort-transactions. Various negotiation protocols such as contract net and multi-stage negotiation can be written by the transaction-based protocol. Negotiation strategies can be written in a logic programming language with the above protocol.

Another kind of communication is applied when a message can be partitioned into sub-messages and their execution plan can be generated. A messages is analyzed and its corresponding processing plan is constructed as a dependency graph. A query in conventional distributed databases is such an example. Synchronization information between sub-messages is attached to each sub-message and controlled by the capsule of each agent.

An answer message can be processed by *global constraints*, a *constraint solver*, or aggregation functions. Global constraints are used for restricting special values embedded in messages. For example, you can see the number of columns and rows in n-queen as unchanged one, and a blackboard as changeable one. A constraint solver is used for evaluation of results. An environment sends them to the related agent if necessary. Depending on their evaluation, the environment decides whether alternative message processing is necessary or not.

When all agent cannot solve a query, the environment send the query to the outer environment, which may be a user, through its capsule.

4.3.3 Proxy Agent

To process negotiation more efficiently, *proxy agents* can be generated by a sender agent. A proxy agent is a copy of the original and has restricted authority which can negotiate with others. As a proxy agent is sent with a message, it works in a neighbor process to the other agent: that is, if some messages are required for the negotiation, communication cost can be reduced by the proxy mechanism.

4.3.4 Concurrency Control

Whether an agent with side effect can process multiple messages or not depends on the ability of the concurrency control of the internal problem solver. For example, if the problem solver is an database management system, its capsule sends multiple messages because the problem solver controls concurrency as one of basic functions. If the problem solver does not have concurrency control, its capsule serialize a transaction (a logical sequence of messages) or the environment replicates the agent process.

4.3.5 Human Interaction

A user can play three roles in *Helios*: an end user, an outermost environment, and a problem solver.

For communication between a user and an agent, a user can give his user model, which corresponds to a common type system and data structures defined in an outermost capsule. Given a user model to an agent, its capsule transforms all messages between the user and the agent.

A user is defined as the outermost environment where there is only one (simple or complex) agent. If an internal agent cannot solve a problem, the problem is thrown out in the outer environment. So, a user receives unsolvable problems finally. If the user returns the answer to the agent, the agent continues to process the suspended message.

Furthermore, a user may be defined also as an agent, that is, a user can process a message sent by its capsule and return its result to teh capsule. This feature helps not only prototyping a system, but also constructing a groupware environment, if multiple users are defined as agents.

Such models make prototyping multi-agent programming in our model easier.

Relations among users and agents are shown in Figure 2.

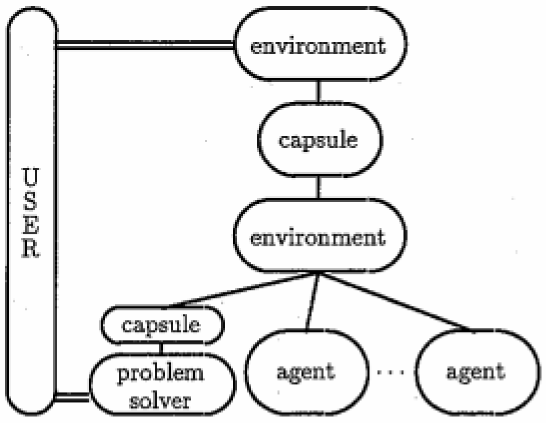

Figure 2: User as an Environment and an Agent

5 What Have Been Achieved?

Here, I summarize what we have achieved by *Kappa*, *QUIXOTE*, and *Helios*: that is, what their contributions are.

Kappa

The major objective of *Kappa* is to store and manage a large amount of structured data and the system had been expected to play a role of an underlying database layer. From this viewpoint, I can list the following contributions:

- We implement a nested relational model efficiently: especially we implement its parallel system[Yokota *et al* 88, Yokota and Yasukawa 92, Kawamura *et al* 92, Kawamura and Kawamura 94].
- We show its efficiency and effectiveness as an underlying database layer in data-intensive applications such as genome databases and natural language dictionaries[Yokota and Yasukawa 92, Tanaka 92].

Considering future works from an application point of view, we must strengthen the aspects of an open architecture, to cope with various applications. For example, user's application programs had to be embedded into *Kappa* to support information retrieval features efficiently in genome databases as in [Kawamura *et al* 92].

QUIXOTE

The major motivation of *QUIXOTE* is to represent and process flexibly various data and knowledge in knowledge information applications. From a database point of view, it is a DOOD language and, from natural language point of view, it is an extension of CIL. The main contributions are as follows:

- We design and implement a deductive object-oriented database language based on subsumption constraints, by which logic and object-orientation concepts are integrated[Yokota and Yasukawa 92, Yokota *et al* 93, Yokota *et al* 94, Yasukawa *et al* 92].
- We show its efficiency and effectiveness for many knowledge information applications such as natural language processing[Tojo and Yasukawa 92, Tojo *et al* 93], genetic information processing[Tanaka 91, Hirosawa *et al* 93], and legal reasoning[Yokota and Shibasaki 93, Nishioka *et al* 94, Takahashi and Yokota 95, Tojo *et al* 95, Tojo *et al* 93].

From experiences on several applications, we must consider many extensions: embedding practical functions such as *math* module explained in [Tsuda and Yokota 94], introducing modality and temporal aspects into the logic, preparing a programming environment, and making a design methodology.

Helios

As seen in future works in *Kappa* and *QUIXOTE*, multiple problem solvers are indispensable for *real* applications, whose combination should be flexible. Further, considering large-scaled applications, we must focus on resource bound environments. Under such backgrounds, we started up a *Helios* project as an indispensable system for many applications. The main contributions are as follows:

- We design and implement a comprehensive testbed for heterogeneous distributed cooperative problem solving, where distributed problem solver and multi-agent based systems can be modeled[Yokota and Aiba 93, Yokota 94, Aiba *et al* 94a].
- We propose a transaction-based protocol for various kinds of negotiations such as contract net, distributed constraint satisfaction, and multi-stage negotiation[Aiba *et al* 94b].

We must investigate its applicability to many applications.

6 Databases and Applications

During investigating various data and knowledge in our applications, we have found common characteristics there as in scientific databases as follows:

- *Massiveness of Data*: There are two kinds of massiveness: large quantity as in astronomical data and weather data; huge value as in genome sequence data and image data.
- *Complexity of Data Structure*: Besides complex objects based on simple data types, there are various kinds of data such as semantic structure in natural language, chemical reactions, coordinates, image, voice, and so on.
- *Ambiguity in data*: As many data are gathered by experiments and observations, there are many erroneous data, which must frequently co-exist even if they are inconsistent.
- *Incompleteness of Data*: As it is very difficult to define all properties of an object concerned, necessary properties might be only in the form of constraints, lacking, or redundant.
- *Variety of Kinds of Data*: There are many kinds of databases even in an application, as in biological databases. As they are mutually related and have common data, their integration is indispensable.
- *Importance of Private Data*: There are many unpublic private data such as experiment results and hypotheses, which must be combined with public ones.

To manage such data, many new features are indispensable as next generation database and knowledge-base systems. The followings are representative:

- *Processing Ambiguity*: For ambiguous data, there are many special features. For example, in biological sequence data, such as homology search for sequence data is required.
- *Features Depending on Applications*: For efficient processing, there are many user-definable features depending on applications, such as special unification and special constraint solver.
- *Thought-Experiment Environment*: It is necessary to correct and revise incomplete information in databases. For the purpose, databases had better provide a thought-experiment environment such as abduction and conditional queries, as in *QUIXOTE*.
- *Processing Time Sequence*: As data obtained by experiments and observations have frequently a property of time sequence, databases should support its relatyed operations.
- *Knowledge Discovery in Databases*: It is very important to abstract or discover new knowledge such as rules from primary data, because primary data is too huge in large-scaled applications. So knowledge discovery in databases is indispensable.
- *User Interface*: As many applications have graphical data such as solid structure and image data, it is indispensable to strengthen graphical user interface.

Through our experiences in databases and knowledge-bases for knowledge information applications, I expect a heterogeneous cooperative knowledge-base system as in Figure 3. In the figure, there are

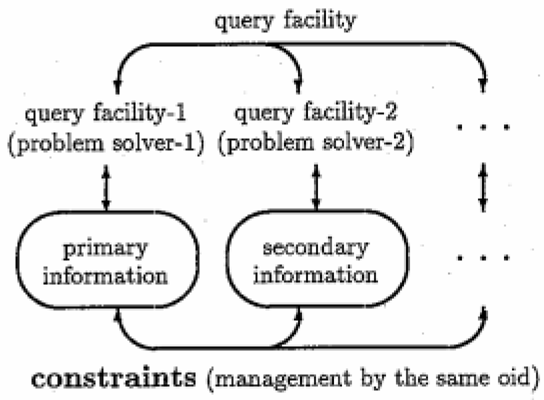

Figure 3: Integrated Management of Heterogeneous Information

two kinds of heterogeneity: abstracted levels and differences of data thmselves. According as such heterogeneity, many problem solvers are required. As they are not separate data, such problem solvers should cooperate to process a query. Such an architecture will be common not only for scientific applications but also multimedia databases, legal applications, and natural language databases. Further, not to mention, we must consider distributed computing environments.

7 Concluding Remarks

Databases have managed conventionally rather clear aspects of the real world and contributed to the productivity of many applications as in business and engineering. However, according as extensions of databases and development of various technologies, we have confronted more difficult applications such as knowledge information applications and scientific databases. We cannot focus only on clear aspects from object domains as in the above applications, because it is difficult to discriminate clear data from 'dirty' ones and we are used to find a diamond in ashes. Further, complex computing environments ask us to combine and utilize various (bound) resources or problem solvers located in distributed environments.

Our works on *Kappa*, *QUIXOTE*, and *Helios*[2] have provided a perspective as in the above, for next generation database and knowledge-base systems.

Acknowledgments

The author is grateful to all members who have and had worked in *Kappa*, *QUIXOTE*, and *Helios* projects, and have discussed their research issues. He would also like to acknowledge Kazuhiro Fuchi and Shunichi Uchida, without whose encouragements the above three projects would not have been succeeded in.

[2]All these software can be obtained by anonymous ftp: ftp.icot.or.jp

References

[Abiteboul and Grumbach 88] Serge Abiteboul and Stephane Grumbach, "COL: A Logic-Based Language for Complex Objects", *Proc. Int. Conf. on Extending Database Technology*, in *LNCS*, 303, Springer, 1988.

[Aiba and Hasegawa 92] Akira Aiba and Ryuzo Hasegawa, "Constraint Logic Programming System: CAL, GDCC, and their Constraint Solvers", *Proc. Int. Conf. on Fifth Generation Computer Systems*, ICOT, Tokyo, June 1-5, 1992.

[Aiba *et al* 88] Akira Aiba, Ko Sakai, Yosuke Sato, David Hawley and Ryuzo Hasegawa, "Constraint Logic Programming Language CAL", *Proc. International Conference on Fifth Generation Computer Systems (FGCS'88)*, ICOT, Tokyo, 1988.

[Aiba *et al* 94a] Akira Aiba, Kazumasa Yokota, and Hiroshi Tsuda, "Heterogeneous Distributed Cooperative Problem Solving System Helios", *Proc. Int. Symp. on FGCS (FGCS'94)*, Dec. 13,14, 1994.

[Aiba *et al* 94b] Akira Aiba, Kazumasa Yokota, and Hiroshi Tsuda, "Heterogeneous Distributed Cooperative Problem Solving System Helios and Its Cooperation Mechanisms", *Proc. FGCS'94 Workshop on Heterogeneous Cooperative Knowledge-Bases*, Dec. 15,16, 1994.

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

[Dadam *et al* 86] Peter Dadam, K. Kuespert, F. Andersen, H. Balnken, R. Erbe, J. Guenauer, V. Lum, Peter Pistor, amd G. Walch, "A DBMS Prototype to Support Extended NF^2 Relations: An Integrated View on Flat Tables and Hierarchies", *Proc. ACM SIGMOD International Conference on the Management of Data (SIGMOD'86)*, 1986.

[Deshpande and Gucht 88] Anand Deshpande and Dirk Van Gucht, "An Implementation for Nested Relational Databases", *Proc. 14th International Conference on Very Large Data Bases (VLDB'88)*, 1988

[Gallaire *et al* 84] Hervè Gallaire, Jack Minker and Lean-Marie Nicolas, "Logic and Databases: A Deductive Approach", *ACM Computing Surveys*, vol.16, no.2, 1984

[Haniuda *et al* 91] Hiromi Haniuda, Yukihiro Abiru, and Nubuyoshi Miyazaki, "PHI: A Deductive Database System", *Proc. IEEE Pacific Rim Conf. on Communication, Computers, and Signal Processing*, May, 1991.

[Hirosawa *et al* 93] Makoto Hirosawa, Reiko Tanaka, and Masato Ishikawa, "Application of Deductive Object-Oriented Knowledge-Base to Genetic Information Processing", Proc. Int. Symp. on Next Generation Database Systems and Their Applications, Fukuoka, Sep., 1993.

[Jaffar and Lassez 87] Jaxon Jaffar and Jean-Louis Lassez, "Constraint Logic Programming", *Proc. the 14th ACM Symposium on Principle of Programming Languages (POPL'87)*, 1987.

[Kawamura *et al* 89] Moto Kawamura, Kazumasa Yokota, and Atsusi Kanaegami, "An Overview of a Knowledge-Base Mamagement System Kappa", *Journal of Japan Society of Artificial Intelligence*, vol.4, no.3, 1989. (in Japanese)

[Kawamura *et al* 92] Moto Kawamura, Hiroyuki Sato, Kazutomo Naganuma, and Kazumasa Yokota, "Parallel Database Management System: *Kappa-P*", *Proc. International Conference on Fifth Generation Computer Systems (FGCS'92)*, ICOT, Tokyo, June 1-5, 1992.

[Kawamura and Kawamura 94] Moto Kawamura and Toru Kawamura, "Parallel Database Management System: *Kappa*", *Proc. International Conference on Fifth Generation Computer Systems (FGCS'94)*, ICOT, Tokyo, Dec. 13,14, 1994.

[Kifer and Lausen 89] Michael Kifer and Georg 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.

[Kiyama and Yokota 88] Minoru Kiyama and Kazumasa Yokota, "Query Evaluation in Nested Deductive Databases", *Proc. SIGDBS of IPSJ*, Mar.15, 1988. (in Japanese)

[Kurozumi 92] Takashi Kurozumi, "Overview of the Ten Years of the FGCS Project", *Proc. International Conference on Fifth Generation Computer Systems (FGCS'92)*, ICOT, Tokyo, June 1-5, 1992.

[Makinouchi 77] Akifumi Makinouchi, "A Consideration on Normal Form of Not-Necessarily-Normalized Relation in the Relational Data Model", *Proc. 3rd International Conference on Very Large Data Bases (VLDB'77)*, 1977.

[Miller 86] Dale Miller, "A Theory of Modules for Logic Programming", *Proc. International Symposium on Logic Programming (SLP'86)*, 1986.

[Miyazaki *et al* 89] Nobuyoshi Miyazaki, Kazumasa Yokota, Hiromi Haniuda, and Hidenori Itoh, "Horn Clause Transformation by Restrictor in Deductive Databases", *Journal of Information Processing*, vol.12, no.3, 1989.

[Mukai 88] Kuniaki 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.

[Nishioka *et al* 93] Toshihiro Nishioka, Ryo Ojima, Hiroshi Tsuda, and Kazumasa Yokota, "Procedural Semantics of a DOOD Programming Language *Quixote*", *Proc. Joint Workshop of SIGDBS of IPSJ and SIGDE of IEICE (EDWIN)*, July 21-23, 1993. (in Japanese)

[Nishioka *et al* 94] Toshihiro Nishioka, Kazumasa Yokota, Chie Takahashi, and Satoshi Tojo, "Constructing a Legal Knowledge-base with Partial Information", *Proc. ECAI'94 Workshop on Artificial Normative Reasoning*, Amsterdam, Aug. 8, 1994.

[Monterio and Porto 89] Luis Monterio and António Porto, "Contextual Logic Programming", *The International Conference on Logic Programming (ICLP'89)*, 1989.

[Monterio and Porto 90] Luis Monterio and António Porto, "A Transformational View of Inheritance in Programming", *The International Conference on Logic Programming (ICLP'90)*, 1990.

[Schk and Weikum 86] HansJörg Schek and G. Weikum, "DASDBS: Concepts and Architecture of a Database System for Advanced Applications", *Tech. Univ. of Darmstadt, TR*, DVSI-1986-T1, 1986

[Scholl and Schk 87] Marc H. Scholl and Hans-Jörg Schek, (eds.), *Theory and Applications of Nested Relations and Complex Objects – An International Workshop, Workshop Material*, 1987

[Starkie 57] Walter Starkie (tr. and ed.), "Don Quixote of La Mancha by Miguel de Cervantes Saavedra", *A Mentor Book*, Macmillan, 1957.

[Takahashi and Yokota 90] Chie Takahashi and Kazumasa Yokota, "A Deductive Database with Hierarchical Structure", *Proc. Joint Workshop of SIGDBS of IPSJ and SIGDE of IEICE*, July, 1990. (in Japanese)

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

[Tanaka 91] Hidetoshi Tanaka, "Protein Function Database as a Deductive and Object-Oriented Database", *Proc. International Conference Database and Expert Applications (DEXA'91)*, Berlin, Aug., 1991.

[Tanaka 92] Hidetoshi Tanaka, "Integrated System for Protein Information Processing", *Proc. International Conference on Fifth Generation Computer Systems (FGCS'92)*, ICOT, Tokyo, June 1-5, 1992.

[Tojo and Yasukawa 92] Satoshi Tojo and Hideki Yasukawa, "Situated Inference of Temporal Information", *Proc. International Conference on Fifth Generation Computer Systems (FGCS'92)*, ICOT, Tokyo, June 1-5, 1992.

[Tojo *et al* 93] Sotoshi Tojo, Hiroshi Tsuda, Hideki Yasukawa, Kazumasa Yokota, and Yukihiro Morita, "*Quixoτε* as a Tool for Natural Language Processing", *Proc. 5th International Conference on Tools with Artificial Intelligence (TAI'93)*, Boston, USA, Nov. 8-11, 1993.

[Tojo *et al* 95] Satoshi Tojo, Stephen T.C. Wong, Katsumi Nitta, and Kazumasa Yokota, "Formalization of Legal Reasoning Based on Situation Theory", *Transactions of Information Processing Society of Japan*, 1995. (in Japanese)

[Tsuda and Yokota 94] Hiroshi Tsuda and Kazumasa Yokota, "A Knowledge Representation Language *Quixoτε*", Proc. Int. Symp. on FGCS (FGCS'94), Dec. 13,14, 1994.

[Tsukamoto *et al* 91] Masahiko Tsukamoto, Shojiro Nishio, and Mitsuhiko Fujio, "DOT: A Term Representation Using DOT Algebra for Knowledge Representation", *Proc. 2nd International Conference on Deductive Object-Oriented Databases (DOOD'91)*, Munich, 1991.

[Uchida *et al* 93] Shunichi Uchida, Ryuzo Hasegawa, Kazumasa Yokota, Takashi Chikayama, Katsumi Nitta, and Akira Aiba, "Outline of the FGCS Follow-on Project", *New Generation Computing*, vol.11, pp. 217-222, 1993.

[Ullman 87] J.D. Ullman, "Database Theory — Past and Future," *Proc. the Sixth ACM Symposium on Principles of Database Systems*, 1987.

[Verso 86] Jules Verso, "VERSO: A Data Base Machine Based on Non 1NF Relations", *INRIA-TR*, 523, 1986

[Yasukawa and Yokota 90] Hideki Yasukawa and Kazumasa Yokota, "Labeled Graphs as Semantics of Objects", *Proc. Joint Workshop of SIGDBS and SIGAI of IPSJ*, Nov., 1990.

[Yasukawa *et al* 92] Hideki Yasukawa , Hiroshi Tsuda, and Kazumasa Yokota, "Objects, Properties, and Modules in *Quixoτε*", *Proc. International Conference on Fifth Generation Computer Systems (FGCS'92)*, ICOT, Tokyo, June 1-5, 1992.

[Yokota 88] Kazumasa 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* 88] Kazumasa Yokota, Moto Kawamura, and Atsusi Kanaegami, "Overview of the Knowledge Base Management System (KAPPA)", *Proc. International Conference on Fifth Generation Computer Systems (FGCS'88)*, ICOT, Tokyo, 1988.

[Yokota and Nishio 89] Kazumasa Yokota and Shojiro Nishio, "Towards Integration of Deductive Databases and Object-Oriented Databases: A Limited Survey", Advanced Database System Symposium, Dec. 7-8, 1989.

[Yokota and Nishio 90] Kazumasa Yokota and Shojiro Nishio, "Deductive and Object-Oriented Databases", *Joho Shori*, vol.31, no.2, 1990. (in Japanese)

[Yokota and Yasukawa 92]
Kazumasa Yokota and Hideki Yasukawa, "Towards an Integrated Knowledge-Base Management System — Overview of R&D on Databases and Knowledge-Bases in the FGCS Project", *Proc. International Conference on Fifth Generation Computer Systems (FGCS'92)*, ICOT, Tokyo, June 1-5, 1992.

[Yokota 92] Kazumasa Yokota, "Knowledge Information Processing in *Quixoτε*", *Proc. SIGDE of IEICE*, May 22, 1992.

[Yokota *et al* 93] Kazumasa Yokota, Hiroshi Tsuda, and Yukihiro Morita, "Specific Features of a Deductive Object-Oriented Database Language *Quixoτε*", *Proc. ACM SIGMOD'93 Workshop on Combining Declarative and Object-Oriented Databases (WCDOOD)*, Washington DC, USA, May 29, 1993.

[Yokota and Shibasaki 93] Kazumasa Yokota and Makoto Shibasaki, "Can Databases Predict Legal Judgments?" *Proc. Joint Workshop of SIGDBS of*

IPSJ and SIGDE of IEICE (EDWIN), July 21-23, 1993. (in Japanese)

[Yokota and Aiba 93] Kazumasa Yokota and Akira Aiba, "A New Framework of Very Large Knowledge Bases", *Knowledge Building and Knowledge Sharing*, eds K. Fuchi and T. Yokoi, Ohmsha and IOS Press, 1994.

[Yokota *et al* 94] Kazumasa Yokota, Toshihiro Nishioka, Hiroshi Tsuda, and Satoshi Tojo, "Query Processing for Partial Information Databases in $\mathcal{Q}\mathcal{U}\mathcal{I}\mathcal{X}\mathcal{O}\mathcal{T}\mathcal{E}$", Proc. 6th IEEE International Conference on Tools with Artificial Intelligence (TAI'94), New Orleans, Nov. 6-9, 1994.

[Yokota 94] Kazumasa Yokota, "Features of Multi-Agent Based Multidatabases", *Proc. Joint Workshop of SIGDBS of IPSJ and SIGDE of IEICE*, July 20-22, 1994. (in Japanese)

[Yokota and Miyazaki 94]
Kazumasa Yokota and Nobuyoshi Miyazaki, *New Database Theory — From Relations to Deductive Object-Oriented Databses*, Kyoritsu-Shuppan, 1994. (in Japanese)

[Zaniolo 88] Carlo Zaniolo, "Design and Implementation of a Logic Based Languages for Data Intensive Applications", *Proc. International Conference and Symposium on Logic Programming (LP'88)*, 1988.