ICOT Technical Memorandum: TM-0056

TM-0056

A Constraint Based Dynamic Semantic Model for Logic Databases

Taizo Miyachi, Susumu Kunifuji
Koichi Furukawa and Hajime Kitakami

April, 1984

ICOT

Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan

(03) 456-3191~5
Telex ICOT J32964

Institute for New Generation Computer Technology

TM-0056

A Constraint Based Dynamic Semantic Model for Logic Databases

T. Miyachi, S. Kunifuji, K. Furukawa and H. Kitakami

Institute for New Generation Computer Technology (ICOT)

ABSTRACT

The utilization of principles and rules in the real world is very important for people and essencial in problem solving. "Constraint" can represent the semantics of this knowledge. This paper discusses the root of descriptive power of constraint and analyzes its four profiles. A database model: "Dynamic Semantic Model for Logic databases" (DSML) is proposed to represent constraint and semantics in the logic databases. DSML can semi-automatically acquire the knowledge about matters which interest the user and can make it easy to manage the life-cycle of logic databases. A DSML database system was also implemented using the declarative logic programming language Prolog.

1. Introduction

It is very important to represent principles and rules in the real world, because they not only represent the real world but give us bases for detecting problems and solving them. 'Constraint' can represent the semantics of the principles and rules, is thus being researched in various areas. Some typical research is an Database system[Ca76],[NG78],[DBG78] Expert system[St80] and Natural language processing[BP83].

Now, in the area of logic databases, research on knowledge assimilation[BK82],[M84] and meta-reasoning[Ki84],[SM84] is being actively conducted. We have much interest in the constraint in knowledge assimilation, dissimilation and accommodation that includes additions, deletions and updates in the databases because that constraint has a close relation with representation of semantics and functions of knowledge in logic databases.

That is, the users have stored the data in the databases after making some data into a relation or relating several data with links on the assumption of existence of some kinds of data. But, as semantics and function in the databases (ex. relationship between more than two kinds of data) have only been embedded in the application programs, it is very difficult to use the data for new purpose or to utilize it for advanced processing of knowledge.

Often, however, the knowledge in the databases is the result of resolution or has possibility of being utilized in the other areas. Therefore, representation of the derivation and semantics related to the knowledge is very important. Constraint can represent them and is very useful for an intelligent computer system which will allow users to utilize this knowledge in the real world. The representation also makes it possible to manage the object real world through logic databases.

In this paper, we will discuss the applying profile of constraint of itself, namely the environments and the mechanisms for applying constraints, in the second chapter. In the third chapter, we will propose a database model named "Dynamic Semantic Model for Logic databases"(DSML) that can represent the constraint of the real world, semantics and function of the knowledge in the logic databases, manage semantics in the databases. In the fourth chapter, we will discuss knowledge acquisition processes that includes knowledge assimilation and dessimilation processes in the logic databases with some simple examples.

2. Applying profiles of constraint

The accumulation of the knowledge appearing in the real world described by an object language is called "Knowledge in the object world" or "Object knowledge". Constraints are able not only to represent principles and rules in the real world that have not been easily represented explicitly, but are also able to activate the useful information related to them fordetermining problems and their solutions.

In this chapter we will investigate about the root of constraint's functions, analyzing their applying profiles.

2.1 Definition of constraint

Changes and actions in the real world can be represented as command sequences in the object world that consisted of definition commands, fetch commands, update commands and delete commands. They are called "Actions." Constraint becomes more useful when the actions happen in general, so it is described with the corresponding actions.

We will propose that "Primitive constraint" is specified in the following form.

```
<Constraint>::=<Pre Conditions>, {<Pre State>:<Pre State Descriptions> ->>
                <Post State>:<Post State Descriptions>}, <Post Conditions>  (A)
```

Here, to specify the image of constraint, we assume that, <Pre Conditions>, <Pre State Descriptions>, <Post State Descriptions> and <Post Conditions> consist of a goal sequence of the logic programming language that has functions equal to or more powerful than those of DEC-10 Prolog. The procedural semantics of <Constraint> is as follows.

"To satisfy <Constraint> for the actions" means the state that the following process sequence is completed for the object world.

1) <Pre Conditions> is satisfied.

2) <Pre State Description> is evaluated only when 1) is satisfied.

3) <Pre State> defined by the evaluation of <Pre State Description> is fetched.

4) <Post State Descriptions> that defines the post state is evaluated.

5) Making <Post State> and substituting <Post State> for <Pre State>.

6) Making sure that <Post Conditions> is satisfied in the new object world.

This is the only state which satisfies <Constraint> for the object world.

Next, we will propose "Action Constraint" ('AC') which can define the necessary conditions of all actions incidental to the primitive <Constraint> in the object world. The syntax of AC is as follows.

```
<AC>::= <Constraint># | <Pre Constraints>, <Constraint> | <Constraint>,
        <Post Constraints> | <Pre Constraints>, <Constraint>, <Post Constraints>
<Pre Constraints> ::= <Pre Constraint> | <Pre Constraint>, <Pre Constraints>
<Post Constraints>::= <Post Constraint> | <Post Constraint>, <Post Constraints>
<Pre Constraint> ::= <AC>
<Post Constraint>::= <AC>                                    ... (B)
```

<Constraint> is that of (A).

Here, <Pre Constraints> is pre-conditions about constraints for the <Constraint> and <Post Constraints> is post-conditions about constraints for the <Constraint>. The procedural semantics of '<AC> is satisfied' is as follows.

The <AC> incidental to the primitive <Constraint> in the object world is satisfied when <Pre Constraints> is satisfied before <Constraint> is satisfied and <Post Conditions> is satisfied after <Constraint> is satisfied. That is the only case which '<AC> is satisfied'.

2.2 Analysis of applying profile of constraint

Analysis of applying profile of constraint is important for the following four points.

a) The necessary items for specifying the real world with its significance become clearer as we the grasp characteristics of constraint.

b) Clues for useful utilization of constraint are obtained.

c) Guiding principle for implementing functions of constraint are acquired.

d) Clues for finding out the cause about the existence of a constraint can be obtained.

The macro-format <AC> defined by (B) is finally expanded into the goal sequence of logic programming language that can be interpreted procedurally. The goals of the sequence are applied in turn, as queries whose form is actions for the object world. We will try to systematize the concept of constraint by analyzing <AC> expanded into the goal sequences.

(1) World Constraint

When <Pre State> to be fetched is always as the same as <Post State> that is results in all the <Constraint> in the <AC>, individual <Constraint> is specified by the following format. That is, (A) is compressed into (C).

```
<Constraint>::= <Pre Conditions>, {<State>:<State Descriptions>},
                                        <Post Conditions>       (C)
```

The states of the object world do not change although the action exists. In this case, <Constraint> is called "World Constraint" ('WC') as it depends on the object world. WC can define the framework of the object world to specify the object world itself as a significant existence and can specify the static or stable state of the object world.

That is to say, WC specifies the necessary condition of the frameworks when the object world is stable. When some action affects the changes in the object world, WC is applied before and after the action happens. That is, 'check of WC' means 'check of the necessary conditions in the object world'. The concept of WC is considered the same as that of 'integrity constraint' discussed in the database theory.

(2) Pure Action Constraint

When the condition mentioned in (1) is not satisfied, this is called "AC is purely satisfied" or "Pure AC ('PAC') is satisfied".

Next, we will analyze this AC from a different aspect.

(a) Transition Constraint

The actions in the object world are substituted into the goal sequence of the proper logic programming language similar to macro-expansions. In the substitution of a pure AC, 'State Transition' generally occurs. In the course of this substitution, ACs are expanded in turn with state transitions. This set of ACs is called "The set in the relationship of Transition Constraint('TrC')." When the cardinality of the set is neither 0 nor 1, a pure AC which is an element of the set is called "Transition Constraint." When the cardinality is 1, AC equals <Constraint> and the AC is called "Individual Constraint" when it is independent from other ACs in the meaning of the transition.

(b) Dependency Constraint

In the course of evaluating the goal sequence acquired with the expansion in (a) using the logic programming language, the values are usually transfered between ACs with common logic variables. These ACs are called "ACs are constraints that depend on each other." In other cases, they are "ACs are constraints that do not depend on each other." We can define the concept of the set that is in the relationship of "Dependency Constraint ('DeC'), and "Independent Constraint," respectively. An element of the former set is called "Dependent Constraint."

(c) Class Constraint

A constraint is sometimes applied for all the elements of some class or some virtual class at the same time. This constraint is called "Class Constraint ('ClC').

ClC is different from the constraints that are applied for each element of the class, because the WCs related to the ClC are applied for the enire class, not for each element of the class.

(Ex. 1) When the salary of a person belonging to a cirtain rank is raised, that of all persons belonging to the same rank should be raised.

(d) Time Constraint

The constraint which defines the time or absolute time when a constraint is applied is called "Time Constraint ('TiC')."

(Ex. 2) Constraint(C2) should be applied after 3 minutes when constraint(C1) is applied.

(Ex. 3) Constraint(C3) should be applied at 8 o'clock every weekday.

2.3 Research on constraints

The relationship between former research and the applying profile of constraint is analyzed in this section. We will also discuss on applying profile that can be implemented using Prolog.

In the research on databases constraints, "Integrity Constraint('IC')", "Integrity Checking" or "Trigger" have been proposed. The research on IC corresponds to that of world constraint(WC), and Nicolas et al.[NG78] researched individual constraints belonging to transition constraints in active constraints. The concept of trigger corresponds to that of time constraints[Ca76]. (refer to Fig. 2.2)

"Dynamic Semantic Model for Logic Databases"('DSML') that we will propose in section 4 can realize all types of constraints mentioned before. But unfortunately, it seems that 'Time Constraint' can not be implemented using Prolog. Because most of Prolog systems do not have the timer.

Applying profile of constraint					Former researches about constraint
WC	pure AC				
	TrC	DeC	ClC	TiC	
o	※	※	※	※	Integrity Constraint (State Law)
※	o	※	※	※	Transition Law
※	※	※	※	o	Trigger
o	o	o	o	o	DSML model
o	o	o	o	※	DSML model implemented by Prolog

Fig. 2.2 The relationship between former research and the applying profile of constraint

3. Semantic representation and constraint in logic databases

In this chapter we will propose how to use constraint to represent the real world in logic databases and manage the part of the real world with logic databases.

First, we will discuss the correspondence between the real world and logic databases.

A thing which exists in the real world is called an "Object." Object changes from hour to hour and arises out of "Action." Object is also action itself and common sense is a object caused from the changes and actions. As semantics and relationships between objects is more useful when changes and actions occur, it is very important to represent the scene, conditions and degrees of importance. We can do such important things using constraint.

Also, in logic databases, data and knowledge generally reflects a part of the real world and its changes. Their existence means the existence of an object in the real world, and the additional-definition, addition, update and deletion in logic databases correspond to actions in the real world.

For example, additional-definition and addition in logic databases corresponding to acquisition of useful information, increase of reserves, discovery of new method and increase of unuseful information etc. The deletion corresponds to disappearance of useful matter, decrease of reserves, abolishment of old method and throwing away of unnecessary information. The update corresponds to the improvement or the worsing of methods and the changes of useful information and reserves.

From the above facts, we can find the possibility of managing part of real world through logic databases.

3.1 Compound world and semantic representation in logic databases

A database usually has several purposes and one world for each one. This world is called "Unit World." A set consisting of one or more unit world is a "Compound World." When one or more unit worlds have the same nature, the compound world is made corresponding to the nature. A compound world represents a partial real world for the nature. A database is organized with compound worlds and represents the real world that the user wants to manage. (Refer to Fig. 3.1)

An object in the real world is called a "Reflected Object (for short, 'RO' or 'R-object') " in logic databases. It is as though users were watching the real world in a mirror. In general, a R-object has different semantics and characteristics in each unit world. Then, as the nature of the unit world depends on the semantics and function of R-object and the relationship between R-objects, a concrete definition of a unit world should be specified using the constraint that defines the necessary conditions for the ROs. We can specify reflectiuons using world constraint(WC) and action constraint(AC).

Only a R-object which satisfies WC is admitted as one which dose not contradict the basic intention of the logic databases. "The R-object satisfies WC" means "not(WC) is not satisfied in the object databaso." ACs accommodate uthe knowledge in the compound world or in the database to prevent contradiction in them, when the R-object wich satisfies WCs is assimilated or obtained.

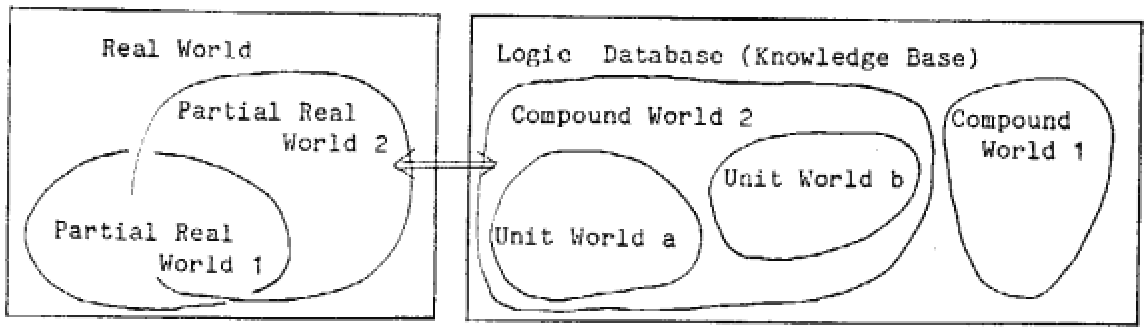

Fig. 3.1 The correspondence between Real World and Logic Database

Essencial actions or accommodations on various scenes can be specified by ACs.

3.2 Dynamic semantic model for logic databases

The database model[Co70],[Ch76],[St80],[HM81] specified using constraint can represent static and dynamic semantics in relations() and in relationships between relations. It is called "Dynamic Semantic Model for Logic databases ('DSML')." Rules are represented using the syntax of Prolog and each of their heads and goals are looked upon as a relation. Using DSML we can represent following three semantics.

1) The stable necessary condition in the compound world can be represented by describing it with world constraint which can specify stable semantics in the relations.
2) The necessary environment and conditions for actions in the unit world or compound world can be represented as user images. The semantics and relationship between relations, for example those of transition, dependency, class and time, can be represented by describing the scenes, actions, sequences of scenes and assignment of acquisition of history with the pure action constraint.
3) The semantics and relationship between the compound worlds in the logic databases can be represented as in 2).

The following four functions make it possible to represent semantics, relationships, changes, etc. in the real world.

a) Meta-knowledge (constraint) can be represented in declarative form.
b) Each constraint can be represented independently and additions of constraint can extend the object partial real world to be managed.
c) Representation of the real world in combination of extensions and intensions as users like can be done. The functions of Prolog interpreter support this to be advanced ways.

(*) Relations may not satisfy the conditions of normal form.

d) The concept of abstraction can be represented using relations, attributes, instances and constraints.

c) means DSML can represent the meanings that can be represented by "semantic network." As Prolog interpreter can be easily implemented using Prolog, the representation of the meaning that is the results of unifications is possible. The reason for 4) is to be able to represent a R-object as any one of a relation, attribute, instance

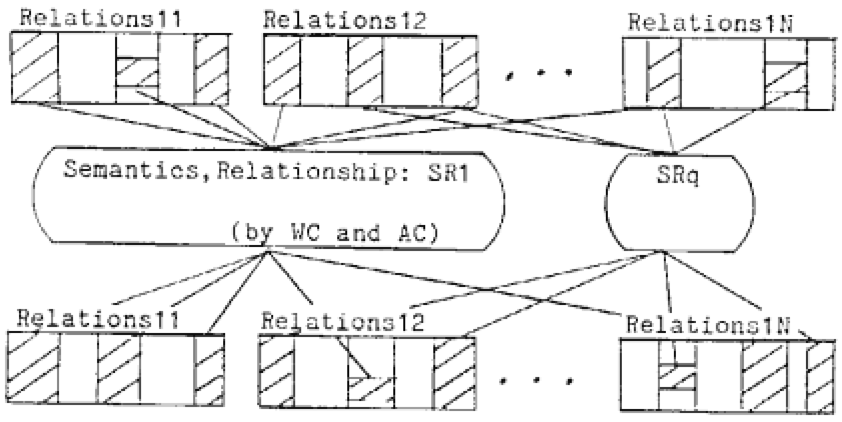

Fig. 3.3 The concept of descriptions of semantics and relationships between relations by DSML.

3.3 Consistency in logic databases

Management to prevent contradictions in logic databases is essential. The consistency can be defined more precisely using DSML under the assumption of closed world assumption('CWA')[R78].

'The database is consistent' means 'not to detect a contradiction in the database' and is the state that satisfies the following condition. demo(W,G) corresponds to "W - G" in logic and means 'G can be derived form W in first order logic'. substitute(A,B) means 'A is substituted by B'.

```
no_contradiction(Compound_world,R-objects) ->
  not(demo(Compound_world,  not(World_Constraint))),
  demo(Finished_Constraints_list, Pre_Constraints),
  demo(Compound_world, Pre_Conditions),
  demo(Compound_world, Pre_State_descriptions),
  demo(Compound_world, Pre_State),
  demo(Compound_world, Post_State_descriptions),
  substitute_state(Compound_World, Pre_State, Post_State),
  demo(Compound_world, Post_Conditions),
    no_contradiction(Compound_world4, Following_Constraints),
    not(demo(Compound_world,  not(World_Constraint))).
```

< Check of consistency >

The constraints to define consistency in the logic databases form the tree for checking itself. Each action constraint described in them is node of the tree. World constraint is applied for each node around the time the constraint assigned by the node is applied. The consistency in the logic database is checked with depth-first parsing of the tree.

Users can view the partial real world by looking over the tree. Users can investigate ways for managing that real world.

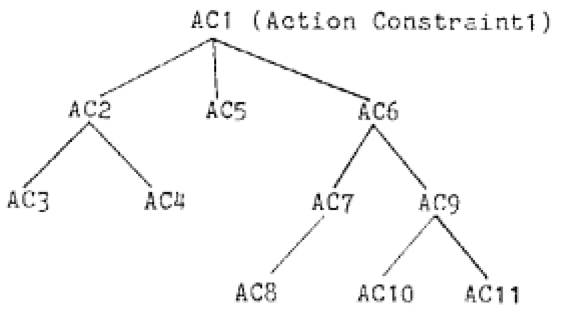

Fig. 3.4 The tree of checking Action Constraints

3.4 Management of knowledge acquisition in logic databases

Input knowledge can be obtained automatically in the logic database in which the semantics of relations and relationships between them are defined using DSML. In this chapter we will explain the process of knowledge acquisition in logic databases. The basic processes of knowledge acquisition are following four sub-processes.

1) New knowledge that causes contradiction in the database is removed.

Because the fact that a new knowledge causes contradiction in the database is detected by applying world constraint, the new knowledge isn't acquired.(Ia)

2) Knowledge in the object compound world are accommodated to make the compound world consistent, after new knowledge is acquired.

More detailed basic processes are the following five subprocesses.

a) New knowledge is acquired in the database and no other action occurs.(Ib)

b) New knowledge is acquired in the compound world and transitive changes occure in it.(Ic)

c) New knowledge is acquired in the compound world and all elements of the class change at that time.(Id)

d) New knowledge is acquired in the compound world and after the assigned time other changes occur in it.

3) Knowledge in the database is accommodated to make the database consistent, after new knowledge is acquired.(Ie)

A more detailed process is just like that of 2).

4) The action occurs at the assigned time.

Fig. 3.5 shows the processes of knowledge acquisition. The boxes out of the database mean a input of new knowledge intended by the user. The nodes in the database correspond to actions, an arc between actions shows the transition of actions. The check of 1) is applied to each node. Here, as knowledge is acquired automatically, the user should only recognize them. Knowledge useful for common sense are automatically acquired.

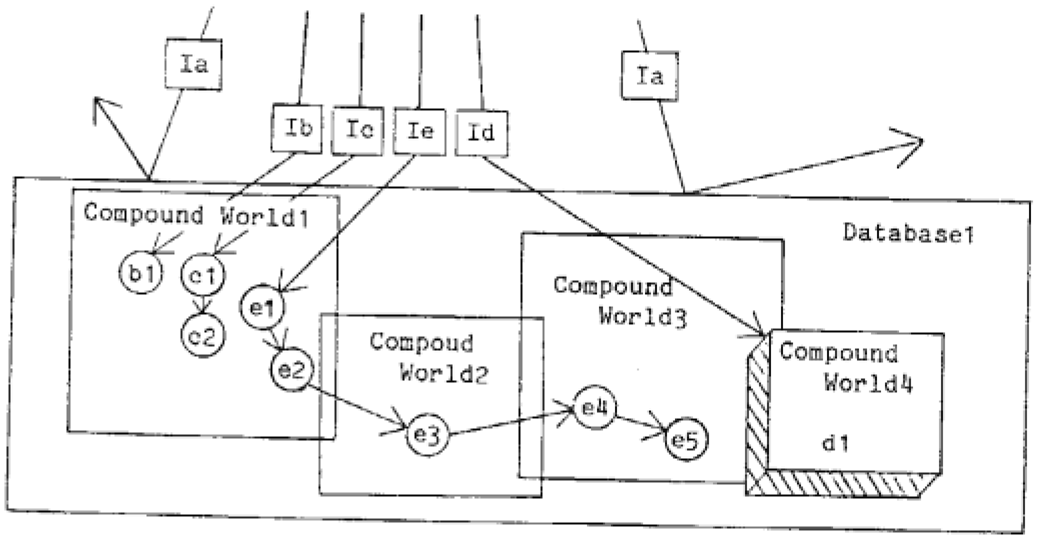

Fig. 3.5 The processes of knowledge acquisition

3.5 Appearance of DSML database systems

In this section we will explain how a DSML database system can be a good assistant for people.

(1) Knowledge acquisition function

As a DSML database system can obtain and manage knowledge following the user's aim and intention, it can be a good assistance for people. That is, the user may only describe the partial real world that he wants to manage, then the logic database system automatically adds, updates or deletes the knowledge in which the user has interest following the user's intention. The knowledge acquired by this function is that which people learn involuntarily or forget to learn and is the base knowledge for fostering common sense.

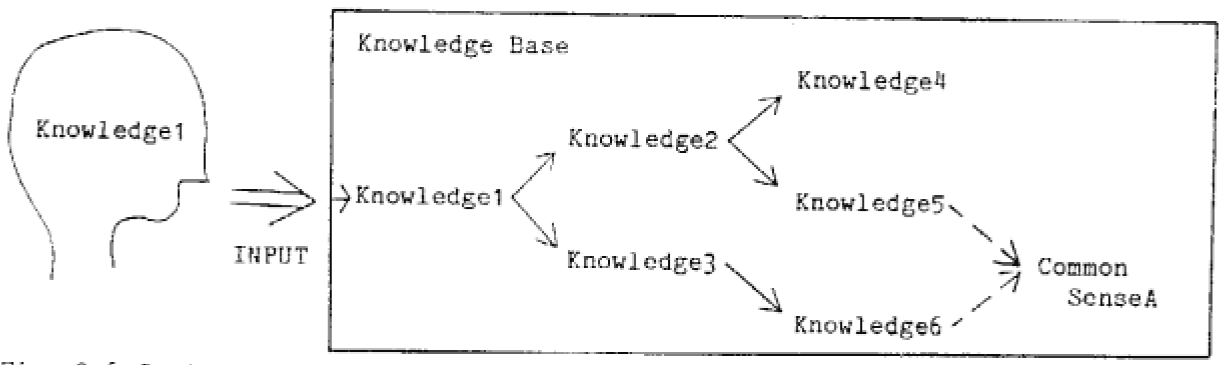

Fig. 3.6 Semi-automatic acquisition of knowledge

(2) Database system design and management of database system life cycle

DSML database systems make it possible for users or database administrators to be able to do database system(DBS) design, DBS management and making application programs at a time with only the database design. (refer to Fig. 3.7)

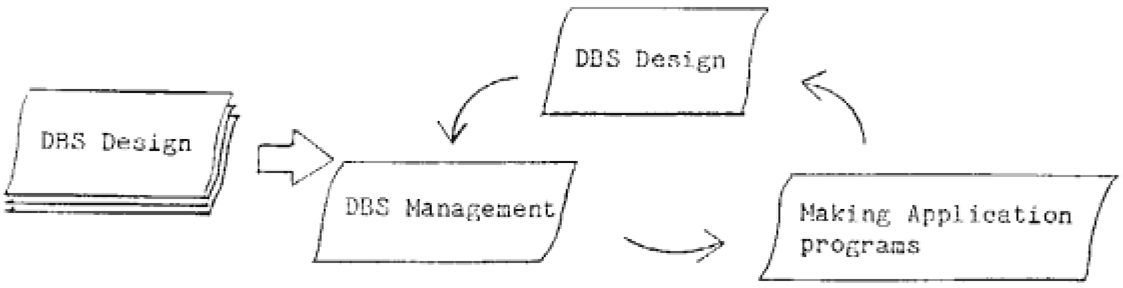

Fig. 3.7 Unifying design and management of DBS and making application programs

Users and DB administrators can view the aim of the DB, semantics and function of knowledge in the DB that is specified using DSML. Thus it becomes easier for users and administrators to actively manage the cycle of the DBS (e.g. revaluation of design and management).

4. Implementation of knowledge acquisition programs

An implementation of logic databases designed by DSML using declarative logic programming language : Prolog is shown in this chapter.

4.1 Syntax for semantics representation

Knowledge is represented in logic databases using Facts(extension), Rules(intension) and constraint. As constraint represents meta level knowledge which specifies semantics of R-objects and relationships between R-objects, it is specified by syntax different from that of object level knowledge (facts,rules).

The following four points are important for represention of constraint.

a) To represent each R-object independently.
b) To specify with declarative forms.
c) Only a small amount of specification is enough to specify constraint.
d) To restrict the kinds of forms for specifying constraint.

Each reason is as follows.

Ra) It becomes easy to add, update and delete constraint.
Rb) It becomes easy to read constraint and to detect errors in it.
Rc) The complexity for specifying constraint decreases.
Rd) The number of rule formats for specifying constraint that the user must remember are few.

4.1.1 Syntax of World Constraint(WC)

World Constraints will be defined by specifying them within the framework of "checkWC", together with the list of object unit world's name, input and message upon detection of contradiction. WC is specified by the format of Horn logic just as is integrity constraint[M 84].

```
checkWC( a list of object unit world's name,  input,  WC,
                  "message upon detection of contradiction" ).
```

4.1.2 The syntax of Action Constraint(AC)

Action Constraints will be defined by specifying them within the framework of "checkAC." (Refer to Fig. 4.1) The precise description in the AC is specified by goal sequences of Prolog.

The first term in checkAC is the identifier of the AC. The second term is input (a new knowledge). Here, users may describe input as "update('Old tupple','New tupple')" to request a update of the knowledge, and may describe it as
"remove(Knowledge)" to dissimilate the knowledge. The third term specifies

a pair that consists of the object compound world specified by the list of unit world and the necessary ACs that have already been applied. The third term also specifies a pair that consists of the object compound world and the prohibited ACs that should not be applied in the world. The transition of scenes that have been checked before the object action occurs can be specified by these conditions for the pre-constraints.

The fourth term is a list that specifies object actions. The following items are specified in it. a) Compound World (a list of unit world's name), b) Attributes that is objects of Class Constraint(list), c) The environment before the object actions occur, d) The state for the actions itself before the actions occurs, e) The state for the actions itself when the actions finish and f) The description to define the values of the state shown by e).

The scene arising out of the actions is specified with the third term, a) and c). b) enables to specify Class Constraint and the attributes assigned by b) is the object of the class constraint. The concrete actions and changes are specified by d) and e). The precise description of actions or changes are given by f). The Time Constraint should be described in this syntax.

The fifth term specifies the sequence of ACs that should be applied after the object actions finish with a list. The sequence of ACs can be represented by the third term and this term.

The sixth term specifies the last state that must be satisfied in the object database. This can check whether appropriate actions and changes have occurred or not.

The seventh term assigns whether the knowledge acquired by the actions is significant or not. The derivation of the new knowledge is left when the important new knowledge is acquired. The syntax of derivation is "sysmemory(ID,Derivation)."

The specification of Dependency Constraint is embedded in the AC with the logic variables. Users can view the set of dependency constraint.

```
checkAC(AC_ID,
        Input,
        [[[World_name1, Necessary_Constraint]|N],
         [[World_name2, Prohibitted_Constraint]|P]]
        [[World_name3, Class_Constraint's_attributes,
          Pre_tate_Descriptions, Pre_State ->>
                                Pos_State, Post_State_Descriptions]|AC],
        [[World_name4, Following_Constraint]|FC],
        [[World_name5, Post_Conditions]|PCD],
        Importance )
```

Fig. 4.1 Syntax of Action Condition

4.2 An example implementation of knowledge acquisition programs

The premise for the knowledge acquisition program comprise the following three conditions.

(1) Input is fact(extension).

(2) Closed World Assumption (CWA) is assumed in the database.

(3) Consistency is defined as shown in the section 3.3.

We can easily implement the knowledge acquisition program using Prolog, because Prolog interpreter can be easily implemented in Prolog and a declarative representation is allowed in Prolog.

4.3 Examples of checking consistencies of knowledge acquisition

We will show simple examples of checking consistencies of knowledge acquisition.

(1) World Constraint

A function to prevent acquisition of new knowledge that contradicts the object compound world is needed.
A new knowledge is " At hospital H, a baby (Yoko) was born to Norio and Yumiko. The hospital registers Yoko as their baby, and makes tests to establish that this is genetically correct." Results of the constraint check reveal that the knowledge is genetically wrong. ("Dr. Gregor Johann Mendel (genetist) says 'No'," a message given on detection of inconsistency. is output.) This shows that no child with blood type B can be born to a couple with blood type A and O. Here the World Constraint is as follows.

```
--- Execution results ---

| ?- assimilate([family],blood_type(youko,b),[parent]).
----- A New knowledge is assimilated !!!

yes
| ?- assimilate([family],father(youko,norio),[parent]).

--- Input conflicts with the integrity constraint !!

    Dr. Gregor Johann Mendel says " NO ! "

   < World Constraint >

checkWC([family],father(X,F),
        (blood_type(F,FT),married(F,M),blood_type(M,MT),
         blood_type(X,BT),genes_match(FT,MT,CBT)-->member(BT,CBT)),
        'Dr. Gregor Johann Mendel says " NO ! "')
```

(2) Class constraint

There is a relation : employee(EMP ,ENAME,SAL,RANK), the knowledge that Mr. Sakai is rank(A) in the database(DB1). The class constraint (C1C2) that defines "When the salary of a employee belonging to rank(A) is raised, that of all employees belonging to the same rank should be raised" also exists in the database. In this case, let us suppose that the new knowledge : "Mr. Sakai has 10 % rise in salary" is generated for the database. Then the salaries of all employees belonging to the same rank are raised to maintain the consistency in the database that is defined by C1C2.

The world of the employee's salary is changed globally. Common sense on the employee's salary changes. For example, the average of the salary and the ratio between different ranks.

```
--- Execution results ---

| ?- assimilate([employees],raise(SAL,emp(EN,k_sakai,researcher,SAL),10),[A]).

--- New Knowledge is  emp(7,k_sakai,researcher,550)
--- New Knowledge is  emp(2,s_shibayama,researcher,440)
--- New Knowledge is  emp(3,h_kondou,researcher,330)
--- AC is employees(76,77,75,checkAC(2,raise(_38,emp(_66,k_sakai,researcher,_38
),10),[[],[[[employees],rank_up(_517,emp(_518,_519,_517,_520),manager_c)]]],[[[e
mployees],[_519],[],[emp(_518,_519,researcher,_520)]->>[emp(_518,_519,researcher
,_521)],[_521 is _520*(100+10)/100]]],[],[[[employees],[true]]],1))

--- Action Constraints are checked !!

   < Class Constraint >

checkAC(2,
        raise(SALt,emp(ENt,Namet,Rankt,SALt),Rate),
        [[],[[[employees],rank_up(Rank,emp(EN,Name,Rank,SAL),manager_c)]]],
        [[[employees], [Name], [],
                  [emp(EN,Name,Rankt,SAL)] ->>
                  [emp(EN,Name,Rankt,New_SAL)],
                  [(New_SAL is SAL * (100 + Rate) /100 )] ]],
        [],
        [[[employees],[ave(employees,SALt,emp(ENt,Namet,Rankt,SALt),AVE),
                                                       AVE < 900 ]]],
        1)
```

(3) Transition Constraint

Let us assume that there is a relation: averagesalary(Rank, Averagesalary) in the database(DB1 using (2)). There is also the transition constraint : "When the rank of a employee is raised to 'Manager of department(MD)' in DB1, his salary should be raised to SMD : "SMD = (Average salary of MD) (Rate in department)."

Representations of the scene in which the knowledge become useful are necessary to specify the semantics of the knowledge. We can discriminate between similar R-objects with them. For example, the scene about a raise in salary is different from the case with promotion(ex. this case) or without promotion(ex. (2)). The description of promotion is essential in this example.

In the above, when the rank of Mr. Sakai is raised, the database management system activates and checks the essential action for maintaining the consistency in the database with its autonomous intention. Users may only ascertain the results just like a teacher.

```
--- Execution results ---

| ?- assimilate([employees],rank_up(Rank,emp(EN,s_yamada,Rank,SAL),md),[A
]).

--- New Knowledge is  emp(1,s_yamada,md,600)
--- New Knowledge is  emp(1,s_yamada,md,840)
--- AC is employees(77,78,76,checkAC(1,raise(600,emp(1,s_yamada,md,600),
20),[[[[employees],rank_up(md,emp(1,s_yamada,md,600),md)]],
[]],[[[employees],[],[],[emp(1,s_yamada,md,600)]->>[emp(1,s_yamada,manage
r_c,840)],[840 is 600*(100+20*2)/100]]],[],[[[employees],[true]]],1))

--- Action Constraints are checked !!
--- AC is employees(78,80,77,checkAC(3,rank_up(sub_leader,emp(1,s_yamada,sub_le
ader,600),md),[],[[[employees],[],[],[emp(1,s_yamada,sub_leader,600)]->>[
emp(1,s_yamada,md,600)],[]]],[[[employees],[raise(600,emp(1,s_yamada,mana
ger_c,600),20)]]],[[[employees],[ave(employees,_539,emp(_540,_541,_542,_539),512
),512<800]]],1))

--- Action Constraints are checked !!
```

<Action Constraint>

```
checkAC(3,
        rank_up(Rank,emp(EN,Name,Rank,SAL),md),
           [],
           [[[employees], [], [],
                     [emp(EN,Name,Rank,SAL)] ->>
                     [emp(EN,Name,md,SAL)],
                     [] ]],
               [[[employees],[raise(SAL,emp(EN,Name,md,SAL),20)]]],
               [[[employees],[ ave(employees,SALt,emp(ENt,Namet,Rankt,SALt),AVE),
                          AVE < 800 ]]],
               1) ).

checkAC(1,
        raise(SAL,emp(EN,Name,Rank,SAL),Rate),
        [[[[employees],rank_up(Rank,emp(EN,Name,Rank,SAL),md)]],[]],
        [[[employees], [], [],
                 [emp(EN,Name,Rank,SAL)] ->>
                 [emp(EN,Name,Rank,New_SAL)],
                 [(New_SAL is SAL * (100 + (Rate * 2)) /100 )] ]],
        [], [], 1)
```

(4) Dependency Constraint and Transition Constraint

Let us suppose Dependency and Transition Constraint : "When the rank of a employee is raised to 'Chief manager of department(MC)' in DB1, his salary should be raised to SMC : "SMC = (Average salary of MC) (Rate in department) and he can receive authority(ex. authority for entering the rooms)" for the DB1. As the second AC (AC2) for authoritycheck1 receives the informaton of the rank, salary and development from the first AC, AC2 is a dependency constraint.

In this case, when the position of Mr. Yamada is raised, the database management system maintain the consistency about the salary and the authority in the database with its autonomous intention.

```
--- Execution results ---

| ?- assimilate([employees],rank_up(Rank,emp(EN,n_yamada,Rank,SAL,DEPT),mc),[A])

--- New Knowledge is  emp(4,n_yamada,mc,1176,investigation)
--- New Knowledge is  authority(mc,4,_1759,investigation)
--- TIC is employees(31,32,30,checkAC(5,authority_check(4,n_yamada,mc),[],[[[emp
loyees],[],[emp(4,n_yamada,mc,1176,investigation),check_authority1(4,n_yamada,mc
,1176,investigation)],[]->>[authority(mc,4,_1759,investigation)],[]]],[],[],1))

--- Transitive Integrity Constraints are checked !!
--- TIC is employees(29,30,28,checkAC(4,rank_up(a,emp(_69,n_yamada,a,_110,invest
igation),mc),[],[[[employees],[],[dept(3,investigation,98),ave_SAL(mc,1200)],[em
p(4,n_yamada,a,700,investigation)]->>[emp(4,n_yamada,mc,1176,investigation)],[11
76 is 1200*98/100]]],[[[employees],[authority_check(4,n_yamada,mc)]]],[],1))
--- Transitive Integrity Constraints are checked !!
```

<Transition Constraint>

```
checkAC(4,
        rank_up(RANK, emp(ENO,ENAME,a,SAL,DEPT),mc),
        [],
        [[[employees], [], [dept(DN, DEPT, DeptRate),ave_SAL(mc,S_MC)],
                  [emp(EN,Name,a,SALpre,DEPT)] ->>
                  [emp(EN,Name,mc,SALpos,DEPT)],
                  [(SALpos is S_MC * DeptRate / 100)] ]],
        [[[employees],[authority_check(EN,Ename,mc)]]],
        [], 1)

checkAC(5,
        authority_check(EN,Ename,mc),
        [],
        [[[employees], [], [emp(EN,Ename,mc,SAL,investigation),
                           check_authority1(EN,Ename,mc,SAL,investigation)],
                  [] ->>
                  [authority(mc,EN,EName,investigation)],[] ]],
        [], [], 1)                                               ... (AC2)
```

5. Conclusions

First of all, we discussed the representaion power and applying profile of "Constraint." We proposed "Action Constraint" and "World Constraint" as general representation forms of constraint and gave their syntax and semantics. The four profiles of constraint that are the transition (Transition Constraint), dependency (Dependency Constraint), class existence (Class Constraint) and time determination (Time Constraint) were shown to be material. We also investigated the relationship between former research about constraint and the four profiles. The constraint is a natural expansion of integrity constraint, essentially.

Next, we proposed a database model : "Dynamic Semantic Model for Logic databases" (DSML) to make database systems better assistants for human knowledge processing. DSML can represent the constraints in the real world as semantics and functions of knowledge in the logic database. The logic database system using DSML enables for the users to manage itself with wonderful easiness. It can allow users not only to use its knowledge but also to manage the object real world.

We implemented the DSML database system by the declarative logic programming language Prolog, and made sure that the concept of constraint is very useful for managing the logic databases and Prolog is suitable for implementing it.

The DSML logic database system can also obtain the knowledge in which users have interest and store the basic knowledge concerning to common sense. We explained that the design and management of the database system and making application programs can be done at a time, and that the management of the life-cycle of the database becomes easy for the database administrator.

Our further research is planned to enhance the descriptive power of the constraint and to implement the time constraint.

Acknowledgements

The authors express their thanks to director Kazuhiro Fuchi of the ICOT Research Center for providing the opportunity to conduct this research.

References

[BBG78] C. Beeri, P.A. Bernstein and N. Goodman; "A Sophsticated Introduction to Data Basa Normalization Theory," Proc. of the 4th VLDB Conf., Berlin, 1978.

[BK82] K.A. Bowen, R.A. Kowalski; "Amalgamating Language and Metalanguage in Logic Programming," Logic Programming (K.L.Clark and S.-A.Taernlund eds.), Academic Press, pp.153-172, 1982.

[BP80] J. Barwise and J. Perry; "Situations and Attitudes," MIT Press, 1983.

[Ca76] J.M. Cadiou; "On Semantic Issues in the Relational Model of Data," Math. Found. Comput. Sci. Mazmkiewiez. Vol.45, Berlin Heidelberg New York , Springer, 1976.

[Co70] E.F. Codd; "A Relational Model of Data for Large Shared Data Banks," Comm. ACM 13,6, pp.377-387, Jun. 1970.

[Ch76] P.P. Chen; "The Entity-Relationship Model-Toward a Unified View of Data," ACM TODS, Vol.1, No.1, Mar. 1976.

[Ki84] H. Kitakami, et al.; "A Methodology of Knowledge Acquisition System," Proceedings of 1984 International Symposium on Logic Programming, Atlantic City, pp.131-142, Feb. 6-9, 1984.

[HM81] M. Hammer and D. McLeod; "Database Description with SDM: A Semantic Database Model," ACM TODS, Vol6, No.3, Sep. 1981.

[M84] T. Miyachi, et al.; "A Knowledge Assimilation Method for Logic Databases," Proceedings of 1984 International Symposium on Logic Programming, Atlantic City, pp.118-125, Feb. 6-9, 1984.

[NG78] J. Nicolas and H.Gallaire; "Data Base: Theory vs. Interpretation," in Logic and Data Bases(H. Gallaire and J.Minker,eds.), Plenum Press, New York London, pp.34-54, 1978.

[R78] R. Reiter; " On Closed World Databases," in Logic and Data Bases (H. Gallaire and J.Minker,eds.), Plenum Press New York London, pp.55-76, 1978.

[Si81] D.W. Shipman; "The Functional Data Model and the Data Language DAPLEX," ACM TODS, Vol6, No.1, Mar. 1981.

[SM84] K. Sakai and T. Miyachi; "Incorporating Naive Negation into Prolog," Proceedings of Logic and Conference, Monash Univ. Jan. 1984.

[St80] G.L. Steele; "The Definition and Implementation of A Computer Programming Language Based on Constraint," MIT AI Labo. AI-TR-595, 1980.