THE CONCEPT OF THE DECLARATIVE INCOMLETE AND FUZZY KNOWLEDGE REPRESENTATION LANGUAGE CONSTRUCTION

SERGEY ASTANIN, VLADIMIR OMELNITSKIY
TAGANROG STATE UNIVERSITY OF RADIOENGINEERING, 44, NEKRASOVSKIY STR., 347928,TAGANROG, RUSSIA
FAX: +7-86344-61787, E-MAIL: SAIT@TSURE.RU

Abstract. The complex systems such as man-machine systems, organizational systems, etc., are often described using incomplete and fuzzy knowledge. We regard the man-machine system as a system of the hybrid intelligence. The basic peculiarity of the hybrid intelligence system is the aspiration to the compromise between interests of the system agents and general purposes of the organization.
We have elaborated the project on creation of computer adaptive intelligent environment (CAIE) for modeling such systems. This project is based on the application of the fuzzy logic methods and plausible reasoning. These methods allow to decide problems of the conceptual analysis (interpretation, diagnostics, prognosis), conceptual synthesis (planning of behavior) and conceptual control (monitoring, dynamic objects control).
We present the declarative knowledge representation language intended for creation of the heterogeneous knowledge base of CAIE. The language includes the following 9 constructions: constant, set, structure, predicate, function, formula, rule, fact, block of knowledge.

INTRODUCTION

The modern concept of artificial intelligence offers the construction of intelligent systems as a net of active agents [1]. Being united in the unified system of the hybrid intelligence, the active agents realize, along with their individual purposes, general purposes of system. We believe that this concept will give the most essential effect at construction of man-machine systems of hybrid intelligence. Such systems are capable to simulate the organizational structures of the most various purpose. The basic peculiarity of the organizational structures is aspiration to the compromise between interests of system agents and general purposes of the organization. The hybrid intelligence represents the joint forecasting mechanism of the organization behavior and functioning. Thus, each agent has partial, fragmentary and incomplete information about the state and changes of the external environment. For hybrid intelligence system the decision is treated as the achievement of some steady states. The decision are realized during exchange of knowledge between agents. It means the acceptance of the coordinated point of view. Proceeding from the nature of interaction between agents of the hybrid intelligence, the software for such class of intelligent systems should be based on logic, plausible and situational models of decisions making. The application of diverse decision making models in the hybrid intelligence systems is based not on agents problem classes only, but the necessity as well to change the logic of reasoning depending on the character and quantity of information both during the interaction and decision searching.

The mathematical methods and decisions making models being a part of the project connected with creation of computer adaptive intelligent environment of the hybrid man-machine systems were selected and developed. These methods are based on fuzzy logic and plausible reasoning and allow to solve the problems of the conceptual analysis (interpretation, diagnostics, prognosis), conceptual synthesis (behavior planning) and conceptual control (monitoring, dynamic objects control). One of the main problems encountering during the system implementation is the knowledge representation, which provides the effective utilization of the decisions making methods and models. To decide this problem the declarative language of incomplete and fuzzy knowledge representation was offered.

When we designed the language the following reasons were took into account:

- The language should know how to describe the structures of knowledge which allow to realize the decisions making models and methods used in the applications of the artificial intelligence. As the analysis shows this purpose is reached if language allows to use such forms of knowledge description as productions, frames, semantic nets and formulas of predicates calculus.
- The structures of knowledge should allow as effectively as possible to realize the conclusion procedures of both logic and not logic type. It's needed to compensate the large volume of tests being the main characteristic of discussed tasks.
- The language should have simple syntax and semantics.

- The language should have open architecture, that is, it should allow to include new constructions without radical revision of the operators already existing.

2 CONCEPT OF HETEROGENEOUS KNOWLEDGE BASE

The dialogue subsystem is one of the base elements of the hybrid intelligence computer adaptive environment. The communication between the agents of the hybrid intelligence is based on the knowledge structures representation language. Such language is necessary as for set-up of computer environment to domains of agents (filling in and updating the knowledge base), and for maintenance of interaction between them during the decision of the common tasks. That's why one of the basic questions is a choice of the problem area description language. The basic requirement to this language is adequacy to objects and relations of the problem area. From the first sight this requirement is fairly simple. There are commercial and experimental expert systems successfully acting in such areas as technical and medical diagnostics, law, multicriterional choice, etc. However almost all systems suffer by one common shortcoming -- orientation to a rather narrow and individual professional problem. Such orientation is often determined by the knowledge representation language chosen for description of the problem area. The expert system development tools also do not rescue this situation because of they have, as a rule, one representation language [2]. All attempts to use these tools in arbitrarily chosen problem area cause the necessity to build either synthetic model of the subject area or update development tools concerning the representation language mainly. The given circumstance is one of the basic obstacles for wide commercial application of the expert systems. We see the decision in development and use the heterogeneous model of knowledge based on various representations. The heterogeneous base of knowledge is a storage of information described various ways. It has more ample opportunities for description of the problem area not limiting the user with one narrow representation. However, the use of heterogeneous knowledge base stays unresolved two following problems: the substantiation of heterogeneous knowledge base structure and the application of the logic and not logic conclusion mechanisms [3]. To resolve these problems two variants are offered.

The first variant means that the heterogeneous knowledge base is a set of probably connected knowledge bases, and each of them is described with its own language. Implementing the given variant, we have to solve problems of the knowledge compilation from one representation to another and development of the powerful user interface, which allows the dialogue in some language not depending on various knowledge representations. Besides, we need some controlling mechanism which selects and launches the necessary logic conclusion mechanisms depending on a current situation.

The second variant assumes development of a uniform declarative type language, which can satisfy the requirements of the diverse logic conclusion mechanisms concerning the representation of knowledge. In this case we need to design one knowledge base only. The controlling mechanism is required as well as in the first variant. It has the following functions:

- Recognition and classification of the input message.
- Allocation of the input message context.
- Analysis of the input message purpose.
- Launch the appropriate logic conclusion mechanism.

As we seem, the second variant of the heterogeneous knowledge base implementation is more preferable since it concentrates all necessary information in one place only.

Nowadays a number of works in this direction are known. Besides, the works, concerning the knowledge representation languages designing, are connected with another scientific direction – the natural languages processing. The basic problem here is absence of the "linguistic-technical" method, which helps the user being not a programmer to create knowledge base. The given circumstance denotes that at the present stage of researches the knowledge representation languages are useful, which have ample opportunities of the subject area description, declarative character of the knowledge representation and syntax formally distinguished from the natural language syntax. Such language will be internal towards the natural one. Furthermore it will be the premise for development of compilers from natural language to its internal representation.

As a benefit of the heterogeneous knowledge representation we consider the fact that bounds of one representation somewhat limit intelligent system. So, logic as the means of knowledge description has definite semantics, definition of the logic conclusion concept and means to analyze the computational and mathematical properties of language. However, there are a number of problems which are not decided by means of "traditional" logic. They are:

- Organization of knowledge when it is easy to determine which units are closely connected with some essence
- Construction of plausible conclusions
- Performing of conclusions which are not certainly correct
- Performing of conclusions on the implicit information in ignorance of the facts

The traditional tools of the intelligent systems construction and use assume that user has sufficient knowledge in programming because of they have advanced languages of queries. They may limit user in the description of problem area on account of menu system limitations. Systems constructed on logical or functional principles (PROLOG-systems, LISP-systems) are, as a rule, even more complex for study and development of the useful intelligent systems. However, the declarative methods are much more preferable because of they have wider functional opportunities, which allow to create intelligent systems.

3 SYNTAX AND SEMANTICS OF THE HETEROGENEOUS KNOWLEDGE REPRESENTATION LANGUAGE

The language is intended for description of knowledge about elements of man-machine system, problems which they solve and purposes of the system by means of construction of hybrid intelligence subject area model.

The language consist of nine statements. They are:

- Constant
- Set
- Structure
- Predicate
- Function
- Formula
- Rule
- Fact
- Block of knowledge

The order of statements in the knowledge base is not fixed. However, there is one restriction: all names used in an operator should be described before their use if they are not formal parameters.

Constant. The constant description has the following form:

constant <name1> = <constant1>,...,<nameN> = <constantN>;

The keyword “constant” defines that the given operator describes constant values in the knowledge base. The <name> is an identifier used further for representation of a constant value. The constants can be numerical, character and logic types. Moreover, descriptions with a degree of confidence are allowed.

Examples (note that the sign // denotes the comment):

Set. The sets are described as follows:

Set <set name> = {<element1>,...,<elementN>};

The sets describe groups of objects, ordinary and fuzzy varieties. They can be used as types for variables and slots of structures. It is allowed to describe empty sets, for example set Empty ={};. For fuzzy set each element has a degree of belonging to this set.

The sets can be modified with the following built - in functions:

```
//Add an element to the set
Set2 = Add(Set1, element)
// Remove an element from the set
Set2 = Remove(Set1, element)
// Unification of sets
Set3 = Union(Set1, Set2)
// Intersection of sets
Set3 = Intersection(Set1, Set2)
```

The set standing at the left side from the sign = can be empty or not, irrespective of this old its contents will be overwritten.

The element of variety can be obtained by its index in the set:

element = Element(Set, index)

The return value depends on type of the set. For fuzzy sets the element is a pair <belonging degree / element value>. For ordinary sets the returned element is a value of the element with passed index.

Except operations listed above the language has function that checks if the element belongs to a set:

result = Belongs (Set, element)

For fuzzy sets this function returns the element belonging degree. For ordinary sets this function returns 1 if the element exists in the set, otherwise it returns 0.

The semantics of the unification and intersection operations depends on a kind of the sets:

1. If initial sets are ordinary then operation of unification is a concatenation of the base sets. The operation of intersection has the usual mathematical meaning in this case.
2. If initial sets are fuzzy then concurrence checking of base sets elements are performed for both operations. The result of two fuzzy sets unification is a fuzzy set of elements received after base sets unification. Besides, the belonging degree for the concurrent elements is calculated as the maximum of their degrees. The result of two fuzzy sets intersection is a fuzzy set of elements received after base sets intersection. And the belonging degree for the concurrent elements is calculated as the minimum of their degrees.
3. If only one of the sets is fuzzy, it is considered that the belonging degrees of the ordinary set are equal to 1.

Structure. The structure is described by means of the following statement:

structure <structure name> =
{
<Slot1>; ...<SlotN>;
};

The description of the structure can be used both for representation of the problem area object classes, and for representation of the particular representatives of these classes. The properties of an object are formalized by means of structure slots. Two kinds of slots exist:

- Built-in slots of the language
- User defined slots

There are two built-in slots of the language. The slot called "Joins" enumerates structures, which the given structure are included in. The second slot called "Includes" describes structures, which join in the given structure as subclasses. These slots allow to represent the structural relations inside of a problem area (namely, the relations "class – subclass" and "class – element").

The user defined slots describe the various characteristics of an object. The form of the slot description in the most general kind looks as follows:

<slot name> : <slot type> = <default value> | <valid values> | <calculating procedure> | <extra procedure>

There are the following slot types:

- Number
- Character
- Boolean
- Predicate
- Rule
- Function
- Set
- User defined type

Depending on the type, the quantity and contents of other fields in the slot description will be vary.

For the numerical, character and logical slot types the default value is a numerical, character or logical constant accordingly. The valid values are a set, whose values can be assigned to the slot. This set can be presented evidently as {<value1> ,..., <valueN>}, otherwise a set described earlier in the base of knowledge is used. The calculating procedure is a name of a predicate (only for the logical type), or name of a function, or number of a rule, which is used to assign a value to the slot. The extra procedure is a name of a predicate, or name of a function, or number of a rule, which is activated if the slot receives a value.

For the predicate, function, rule and set slot types the default value is a name of a predicate, name of a function, number of a rule or name of a set accordingly. The valid values have the same meaning as for the types discussed above, but certainly they have the corresponding type. The meaning of the calculating procedure and the extra one doesn't change.

The user defined type is a name of either a structure or a set described earlier in the knowledge base. The meanings of the other fields are the same.

For all considered cases the slot name and type are not optional only. All other fields may be omitted. For example, the line "WheelNumber : number;" describes the numerical slot, which has not the default value and restrictions on receiving values, besides it obtains the value by means of the direct assignment and it doesn't activate any actions.

If any parts of the slot description are not given, but parts succeeded by them are necessary, then first must be presented with the special sign “?”, as in the following example: Color: ColorSet = ? | ? | GetColor | 50.

This slot receives values from the set of colours, has not the default value and restrictions inside of the base set. It’s calculated with the function GetColor, and after calculation the attempt to prove the rule number 50 is performed.

But the last parts in the slot description may be omitted without restrictions. This is the same example without a rule number: Color: ColorSet =? |? | GetColor;

Predicate. The description of a predicate has the following form:

predicate <predicate name> (<arg1>, ..., <argN>) = <formula>;

The predicates are used for representation of relations between objects of a problem area. The predicate with N arguments sets some relation between N objects. It is possible to describe the predicate with N = 0. In this case the list of arguments in the round brackets is omitted.

The formula is written according to the logic of predicates rules. It can contain another predicates, variables, constants, functions, which are combined with the logical or relational operations. The following table shows the priorities of the operations (smaller digit corresponds to the higher priority):

Priority	Meaning	Notation
1	negation	~
2	equal	==
2	not equal	<>
2	greater then	>
2	less then	<
2	greater then or equal to	>=
2	less than or equal to	<=
3	conjunction	&
3	disjunction	|
3	implication	=>

The round brackets are used to specify the order of the formula evaluation in necessary cases.

The predicate may not contain a body, that is, it simply describes the fact, that there is some relation between objects given as its arguments. In this case the following form is used:

predicate <predicate name> (<arg1>, ..., <argN>);

Examples:

```
//The predicate is interpreted as follows:
//Somebody has the parents, if there is a person in set of the people, which is ether //his mother or father
predicate HasParents(Somebody) = exist x from People Mother(x, Somebody) | Father(x, Somebody);
// use of the generality quantifier
predicate GreatAchievement = all x from Students HasRating(x, “perfectly”);
// abstract example
predicate P(x, y) = ~(P1(x) & P2(y) | P3(x, y)) => ~x;
// the fact
predicate Loves(Andrey, Natali);
```

Formula. The description of the formula matches with the description of the right part of the predicate.

Function. The functions are used for representation of functional dependencies between objects. They are represented as follows:

function <function name> (<arg1>, ..., <argN>) = <expression>;

The expression can contain constants, variables and calls to functions. It’s built using the standard mathematical operations *, /, +, -. To specify the order of the function evaluation the round brackets are used. The infix notation of the operators writing is applied.

Examples:

```
function Square(x) = x * x;
function GetLoadCapacity() = GetCapacityFromDatabase();
```

Rule. To describe a rule the following statement is used:

rule <number> = if <premises list> then <conclusions list> ;

The rules describe relations between objects of a problem area in the terms “cause” and “consequence”. The premises list is an ordinary formula. The conclusions list is also a formula, but which has some restrictions on used operations and operands. Particularly, the conclusions list looks as follows: <conclusion1> & < conclusion2> & ... & < conclusionN>, where < conclusion i > is either a fact or a predicate or a special procedure.

There are three special procedures, which are presented in the following table:

Name	Action
AddFact(F)	Adds the fact F in the knowledge base. The new fact is represented by a predicate.
RemoveFact(F)	Removes the fact F from the knowledge base.
Activate(N)	Indicates the rule which will be executed next.

If the conclusion represents a predicate then the attempt to prove it is done.

Example:

```
// Absolutely abstract example
rule 5 = if
      P(x) & U(y) |
      Func(x) > 100 |
      Parameters.CurrentTime < HiRange
then
      Parameters.Status = “Fine” / 0.85 &
      Activate(15);
```

Fact. The fact description has the following form:

<name> = <expression>

The name is a variable or a structure slot name. To set the slot value, the complex slot name is used. It has the following structure:

<structure name>.<slot name>

The expression must have the appropriate type.

Example:

```
PersonalCars.WheelNumber = 4;
Variant = "exact";
```

Besides, the facts can be described by means of the predicates as follows:

```
predicate man(Bill);
predicate PingpongTeam(Mark, Julia);
```

Knowledge block. The block description is intended for allocation of logically or semantically connected fragments of the knowledge base towards the search and conclusion performance optimization. It has the following form:

block <keys list> {<statements list>}

The parameter <keys list> consists of variables delimited with the commas. If for some variable there is a fact “variable=value” then the contents of the block is used during the logical conclusion. If all keys are failed then another block is considered. The <statements list> is a really fragment of the knowledge base.

CONCLUSION

The considered language allows to describe the heterogeneous knowledge structures for man-machine system behavior modeling. Currently the translator from discussed language to its internal representation has been elaborated. We continue to research the fuzzy conclusions methods and algorithms of plausible reasoning. Simultaneously, we design the decision machine, which is based on the interpretation of the considered language internal representation by means of the fuzzy and plausible reasoning mentioned above application.

REFERENCES

1. Vendler Z. Linguistics in Philosophy. N.Y., 1967.
2. Ohsuga S. Toward intelligent CAD systems//Computer Aided Design. 1989. V21. ¹5.
3. Christaller, Th., Di Primio, F., Vob, A.(eds), Tne AI Workbench BABYLON, Academic Press. 1992.