PROLOG LANGUAGE APPLICATION FOR ALARM SYSTEM REALIZATION IN ACCELERATOR CONTROL

I.Frolov^, A.Vaguine
Russian Academy of Sciences Moscow Radio-Technical Institute (MRTI), Warshawskoe shosse, 132, 113519 Moscow, Russia

I.Abe, K.Nakahara, K.Furukawa, N.Kamikubota
National Laboratory for High Energy Physics (KEK), 1-1 Oho, Tsukuba, Ibaraki 305, Japan

ABSTRACT

Such PROLOG features as backtracking, matching and recursive data representation are powerful tools for ALARM system realization. Although the main idea is the possibility to describe some technical system in recursive form, backtracking and matching are ideal for processing recursive data structures. This paper represents a technique which would allow PROLOG language application for ALARM system realization using an example of the KEK LINAC magnet system. The technique is based on an object-oriented internal data representation in terms of objects, properties, relations and knowledge conception. In addition, each property value is characterized by a typical "time life".

INTRODUCTION

Today's concept of object-oriented programming (OOP) has traditionally been associated with such languages as a Smalltalk, C++, Object Pascal, Ada and Common Lisp Object System. The fact that all of this languages are algorithmic imposes common features on the style of object-oriented programming for these languages as well as realization methods for object-oriented design /1/. In turn, declarative principles of programming allow us to introduce new possibilities as well as a new content in the conception of OOP. We can now speak about a new content of OOP since we understand OOP as being a subject of investigations and development. This statement is based on the fact that an object conception is one of the main philosophic categories - 'Object is all that can be in a relation or has some property' /4/. The current OOP conceptions are poor copies from current philosophical doctrines concerning this point. However, from another point of view, further development of the OOP conception can enrich the philosophical thought. 'Artificial Intelligence (AI) is an experimental philosophy' /3/ and object conception is an integral part of AI. We think more correctly to speak about different generations and about different levels of OOP. In addition, the difference between a real object and his reflection in our consciousness motivates the basis for two different OOP approaches.

In accordance with the aforesaid, we propose to preserve only the first part of the OOP definition in /1/ as the OOP definition in the following form:

Definition: Object-oriented programming is a method of implementation in which programs are organized as cooperative collections of objects.

One of the most powerful declarative languages is PROLOG. The conception of object-oriented programming involving the PROLOG language (OOPP) is presently in the stage of development. The main effort regarding OOPP is gaining an understanding of the role and importance of the internal data representation structure (IDR). In fact, the structure of the internal data representation defines the 'genetic' of the control system. Structure is a basis of cognition, and the most powerful structure is a network structure of data representation. For example, one of the possible network form is a form of IDR in terms of the object, property, relation, role and process. This conception presently includes such ideas as a knowledge equation, virtual object, system consciousness level, intelligent unity, problem solver, goal world, real world, planning world and learning. The integration of all these ideas into one system requires powerful multi-processor supercomputers. However, some separate methods can be successfully applied for solving different problems on ordinary personal computers. Below we consider the principles of some approach of object-oriented programming in the PROLOG language (OOPP), and demonstrate an application of this approach for ALARM system realization based on the example of the KEK (National Laboratory for High Energy Physics, JAPAN) LINAC Magnet System.

OBJECT-ORIENTED APPROACH

Briefly, our approach is as follows:

1. The world consist of Objects (complex and simple, i.e. every object can be part of another object while simultaneously consisting of other objects).
2. Every Object is characterized by set of properties.
3. Each property value has a typical "time life". This is the time at which the property value can be changed significantly (time flies all become different) (for accelerator subsystems it can mean a polling frequency).

---

^ In process of paper preparation, I.Frolov worked in KEK as a foreign researcher.

4. Each pair of Objects can be associated by a proper relation (for example: an injector is a part of an accelerator (in this case, "part of" is a relation).
5. In accordance with the property's values and relations, all Objects can be divided into the classes.

We can represent this in PROLOG terms in the following manner:
Each object can be identified by some identification number *(ID)*: for example, integer. A property can be represented by a predicate with two arguments: identification number and property value. This predicate can work in two different regimes. The first regime is when a property value is a variable:

```
current(1,X):- var(X),..
```

This is a form of request. It means that we want to know the value of the 'current' property. The second regime is for a 'current' predicate which can be presented, for example, in the following manner:

```
current(1,5).
```

This predicate means that the current value of some magnet *(ID=1)* must be equal to *5* (for example, amperes). In this case, the predicate must implement some procedure in order to make the current value for the magnet *(ID=1)* equal to *5* amperes.

The relation between two objects can be represented by a predicate with two arguments: the identificator of the first object and the identificator of the second object. For example, if some accelerator (object) has *ID=1*, and its injector has *ID=2*, the predicate,

```
part_of(2,1).
```

means that the injector *(ID=2)* is a part of the accelerator *(ID=1)*. It's necessary to say here that conceptions concerning property and relation are in principle different. To explain this difference, we introduce the following definition:
Definition: Property is a characteristic of an object which must be preserved under any changes of the surrounding world.
For example, although the mass of some object would be changed under World changes in accordance with Einstein's theory, such characteristics as mass would be preserved. In turn, such a relation as 'president' for M. Gorbachev disappeared (for Gorbachev) after the destruction of the USSR.

To realize the 'time life' conception, we can apply the following approach: we can introduce for each property a database. For example, for a current property we can introduce a database (facts in PROLOG terminology) according to the following form:

```
current_base(ID,Val,TD(Time,Date))
```

Here, *ID* is an object identificator; *Value* is a 'current' value; and *Time* and *Date* represent when this property value was calculated (received). In such a case, the first predicate can be presented in the following manner:

```
current(Id,X):-   var(X),
current_base(ID,X,T_D(Time,Date)),
system_date(CurDate),
system_time(CurTime),
check(Time,Date,CurDate,CurTime).
```

Here the 'check' predicate checks the possibility to use an old property value. In such a case, the following 'current' predicate must write a new current value in the database. For example:

```
current(Id,X):-   var(X),
     camac_address(Id,Address),
     camac_request(Address,X),
     system_date(Date),
     system_time(Time),
     retract(current_base(Id,_,_)),
     asserta(current_base(Id,X,
          T_D(Time,Date)).
```

This predicate reads the current value from a camac device and writes this value into the database. Of course, a database service can be realized in a more sophisticated form: for example, as a stack with a limited depth, or the predicate can control the size of the available memory before inserting a new fact.

About classes:
Definition: 'By a class is usually meant a collection of individuals, to each of which a particular name or description can be applied;' /2/
In turn, in the context of the object-oriented design in /1/ was proposed the following definition:
Definition: 'A class is a set of objects that share a common structure and a common behavior.'
As can be seen, any set of objects is defined in terms of our approach to fits any definition. For example, consider the following set of objects:

```
type(Id,Type),Type = magnet,
sector(Id,Sector),Sector =:= 2.
```

This set is a class of magnets - sector number 2. From one side, this collection can be characterized by such common descriptions as *'magnet'* and *'sector 2'*. From another side, such properties as the type and sector are elements of structures. The common behavior is based on common predicates. For example, *the* *current(...)* predicate is valid for each element of the aforesaid class.

In addition, this approach allows an interesting interpretation of such object notions as a state, behavior and identity. The state of an object encompasses all of the properties of the object, plus the current values of each of these properties; object behavior is expressed in terms of its state changes. In fact, this is in full accordance with the appropriate definitions given in /1/. As for the Identity mechanism, this approach enables us to identify objects by a combination of properties and relations. For example, to change the 'current' value of the 'STO-01X' magnet we can use the following approach:

```
name_(X,'STO-01X'),current(X,5).
```

In this example X is a variable. As can be seen, in order to indicate some object, we don't need to use an ID number in an explicit form. The first predicate - *name(X,'STO-01X')* plays the role of an indicator. However, to indicate the same magnet, we can also use another form - for example:

```
sector(X,1),cntl(X,3),ch(X,1),
current(X,Current).
```

MAGNET SYSTEM

OBJECT-ORIENTED ANALYSIS

The KEK Linac magnet system comprises approximately 360 magnets. In turn, each magnet includes some subset of the following set of subsystems:

- power control (power ON or OFF);
- magnet water control;
- magnet temperature control;
- cooling system of magnet control box (FAN);
- over voltage control;
- over current control;
- transistor temperature control; and
- transistor state control (fault or normal).

One of the possible solutions represents each subsystem as an object on the software level, is characterized by the state property (fault or normal) and is connected to the magnet object by the 'part_of' relation. Another solution is to code the available control subsystems and the current state of this subsystems in bitmask form (for example: bitmask and interlock properties, respectively).

OBJECT-ORIENTED DESIGN

The goal of our system (ALARM system) determines the possibility of an abstraction technique. To successfully realize our goal we don't need to take into account all information concerning the magnets (for example: magnet current and magnet type (steering, bending, quadruple, focusing or virtual)).

In addition, since 360 is a sufficiently big number for human brains, we can apply an object-oriented decomposition technique. In accordance with magnet scheme, all of the magnets can be divided both functionally and geographically into 7 sectors (0,1,2,3,4,5,7). In turn, each sector would contain a few subsystems: power supply controllers. Each power supply controller would be characterized by some number *(cntl)*. In accordance with this structure, we introduce the following additional objects:

Magnet Control System (MCS) (ID = 600)

This object is characterized by the state property (normal or fault) and type property:
type(600,'magnet system').

MCS - sector 0 (ID = 500)
type(500,'magnet system')

....

MCS - sector 7 (ID = 507)

Each object of this group is characterized by the state and sector properties.

MCS - sector 0, cntl 1 (ID = 401)

...

MCS - sector 9, cntl 8 (ID = 498)

Each object of this group is characterized by the state, sector and cntl properties.

OBJECT-ORIENTED PROGRAMMING

For ALARM system realization we shall use only the following magnet characteristics: name of magnet *(name_)*, *type* of object (in our case either magnet or 'magnet system'), state of magnet *(state* (for example normal of fault)), bit mask characterizing the available magnet characteristics under control *(interlock)*, bit mask characterizing the state of some magnet characteristics under control *(bitmask)*, sector number *(sector)*, and power supply station number *(cntl)*, power supply channel number *(ch)*. In the PROLOG form:

```
name(1,'STO-01X'),type(1,magnet),
sector(1,0),interlock(1,40),
cntl(1,3),ch(1,1)
```

This information can be stored as facts.

In addition, the relationship between this object and the Magnet Control System (MCS) sector number (0) and the power supply controller number (1) (MSC-sector0-cntl1) (ID=401) can be represented as a predicate

```
part_of(1,401)
```

In turn, object (ID=401) is a part of the object MCS-sector0 (ID=500), and object (ID=500) is a part of the object Magnet Control System (ID=600). These facts can be represented in the following manner:

```
part_of(401,500)
part_of(500,600)
```

and so on.
We can now introduce the following predicates:

```
bitmask(X,Maskvalue):-
    var(Maskvalue),
    name_(X,MagnetName),
    mgsts(MagnetName,Maskvalue).
```

The mgsts predicate was developed in C language and implements the hardware interface to the magnet equipment.

```
state(X,State):-  var(State),
    type(X,magnet),
    interlock(X,Val1),
    bitmask(X,Val2),
    Val1=Val2,
    State = normal,!.
state(X,State):-  var(State),
    type(X,magnet),
    State = fault,!.

state(X,State):-  var(State),
    type(X,'magnet system'),
    var(StateValue),
    part_of(Y,X),
    not(state(Y,normal)),
    State = fault,!.
state(X,State):-  var(State),
    type(X,'magnet system'),
    State = normal.
```

This predicate means that the state of some object is at fault if the state of some its subparts is at fault; in the opposite case its state is normal.
In order to start up all of the control procedures, we can introduce the following predicates:

```
main_control:-
    repeat,
    state(600,X),
    state_processing(X),
    fail.

state_processing(normal):- !.
state_processing(fault):-
                inform_operator.
```

Although we used ID number (600) in the main_control predicate in an explicit form, we can easily rewrite the state(600,X) predicate in the following form:

```
name_(Y,'Magnet Control System'),
state(Y,X),
```

For this approach realization we used a personal computer (PACKARD BELL (extended memory - 8 Mb, 50 Mhz)) as well as the IF/PROLOG version 4.1.9 MS-DOS 386-486. To poll all magnets, the PROLOG part (the main part) of the program spends approximately 0.66 sec. It's a very good time characteristic for slow control systems. Another advantage of this approach is the simplicity of program development and implementation. In fact, one person can develop such a program in only three weeks. We don't mention here the simplicity of program changes, since it's obvious due to the fact that it is based on the declarative programming principles.

CONCLUSION

This paper presents a brief description of our approach and problem solution. However, we hope that the aforesaid example provides some understanding of the advantages of declarative programming in object-oriented programming. The main advantage is an object Identity mechanism. In comparison with algorithmic object-oriented languages, the Identity mechanism of this approach is perfect. This mechanism opens up the possibility of using a limited natural language and the concept of a virtual object. Acceptable time characteristics and approach simplicity provide good possibilities for practical applications. In addition, today, this approach also includes such conceptions as a 'knowledge equation', system 'consciousness level', problem solver, goal world, real world and planning world. In turn, 'knowledge equation' provides a good tools for the realization of learning mechanisms.

REFERENCES

[1] Booch, Grady - 1991
Object oriented design with applications.
Redwood City: Benjamin/Cummings
[2] George Boole
An Investigation of the Laws of Thought on which are founded the Mathematical Theories of Logic and Probabilities.
Dover Publications, INC., New York
[3] A Future of Artificial Intelligence.
Izdatelstvo Nauka 1991
[4] Philosophy Dictionary.
Izdatelstvo Nauka 1989