TR-024

A Knowledge Representation System in Prolog

by
Kenji Sugiyama, Masayuki Kamcda,
Kouji Akiyama, Akifumi Makinouchi
(Fujitsu Ltd.)

August, 1983

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

Technical Report of ICOT Research Center : TR-0024 30-August, 1983

A Knowledge Representation System in Prolog

Kenji Sugiyama, Masayuki Kameda,
Kouji Akiyama, Akifumi Makinouchi

(Fujitsu Ltd.)

30-August, 1983

ICOT Research Center
Institute for New Generation Computer Technology

Mita Kokusai Bldg. 21F, 1-4-28 Mita,
Minato-ku, Tokyo, 108
Tel:03-456-3195, Telex:ICOT J32964

A Knowledge Representation System in Prolog

Kenji Sugiyama, Masayuki Kameda, Kouji Akiyama, Akifumi Makinouchi

(Fujitsu Ltd.)

1015, Kamikodanaka Nakahara-ku, Kawasaki, 211, Japan

* This work was sponsord by ICOT(Institute for New Generation Computer Technology).

Abstract

This paper presents a knowledge representation system which has been developed as a first step to transport an interactive natural language programming system in LISP environment into PROLOG environment. The representation system is based on frame representation language and is extended by the idea of object-oriented programming. In this system the knowledge of a frame consists of attributes and methods, and a piece of knowledge is represented as a PROLOG clause. Hierachy of the frames is twofold : class/instance and super/sub relations. According through the former relation, frames are classified into three levels : individual, pure-class, meta-class. This object-oriented frame system is more suitable for modeling the task domain of an application system, such as the natural language programming system, than the already proposed knowledge representation systems in PROLOG.

1. Introduction

A number of papers presented an interactive natural language programming system (KIPS) which was developed in Lisp environment. From Japanese specification, KIPS has successfully produced several dozens of programs composed of highly standardized file manipulating operations in business applications. The system, however, lacks the theoretical base for meaning representation, so that the representation dangles in an ad hoc manner. A theory for semantics is needed to extend the acceptable sentences. The LUNAR system showed one answer to this kind of problem. It used a kind of logic. Warren et al. showed the inadequency of the LUNAR treatment and introduced Prolog subset of logic to compensate for it. The approaches taken by these forerunners drew our attentions to the usefullness of Prolog as a basis for meaning representation. This motivated us to transport the current version of KIPS into Prolog environment as a preparation for a semantic theory development.

The first step we have taken is the development of a knowledge representation system in Prolog environment, because it consists of the most indispensable part of the natural language programming system. This paper presents a knowledge representation system which has been developed in Prolog environment. Chapter 2 reviews how KIPS uses knowledge represetation. Chapter 3 describes the developed knowledge representation system (FLORES: Frame-like Logical Object Representation System). Finally, chapter 4 compares FLORES and other already proposed knowledge representation systems in Prolog.

2. Knowledge Representation in KIPS

The knowledges of KIPS are organized around the models of the task domain. One of them is the program model. Fig. 1 shows the model. It is divided into model of processes and that of files, each of which are classified by super-sub relations. The program being specified by a user is represented as objects at instance level. Each objects are realized as frames in KIPS.

The knowledge of building up a user's program model from fragmentary specification is stored in frames at class level. The building processes are done by the demons in frames and message passing like that of LOOPS, as shown in fig. 1. These processes use a knowledge representation system which is based on PRL and is extended by message passing functions.

Current version of KIPS has only knowledge at instance and class levels. But Sugiyama et al. argued that meta knowledge was required to make KIPS more flexible, in such case as to allow a user to use user-defined operations.

3. FLORES

Our endeavour lies in the development of a knowledge representation system in Prolog environment, which is capable of representing such processes as stated in Chapter 2.

3.1 Objectives

The objectives of FLORES are 1) implementing frame reprentation system making use of Prolog features, 2) realizing frames as objects which has two aspects : data (called attributes in this paper) and procedures (called methods). Method represents dynamic feature or behavior of a object, while demon is secondary procedure attached to a declarative datum

(attribute). Achieving 1) and 2) yields a framework for both Prolog-based knowledge representation and object-oriented programming whose characteristic is message-passing.

3.2 Frame Structure

To realize a frame system in Prolog, it is important to consider how to represent frame structure (slot-facet-datum) in Prolog. In Lisp, it is realized as property lists. Considering data and procedures are represented identically in Prolog, a piece of information of frame was determined to be realized as Prolog clause, as shown in fig. 2, because frame has two aspects of data and procedures. You may consider that the term beginning with slot is internal predicate term within the 'world' specified by frame-name (see chapter 4).

3.3 Basic Features

1) hierarchy -- class/instance and super/sub

Hierarchy at FLORES is a extension of AKO hierarchy of FRL, and is classified into class/instance hierarchy and super/sub hierarchy. Class/instance hierarchy can be viewed as three levels as shown in fig. 3.

A frame in class level (meta-class or pure-class level) contains the common knowledge among the instance frames belonging to it. Basically, frames at pure-class level are used to model a task domain of an application system. Individual-level frames are instances of these. Frames at meta-class level contain meta-knowledge which deal with class-level frames.

Super/sub hierarchy has property inheritance, like AKO/INSTANCE of FRL,

2) Types of knowledge -- attribute and method

Types of knowledge in FLORES are twofold: attribute and method, each of which are descriminated by their facet names: $value and $method respectively (fig. 2). Each instance frame contains concrete attribute values. A method in a class frame, which is common in instance frames, does different things according to each different attribute values in instance frames.

The following notations are typically used for retrieving attribute values and activating methods (by message passing).

get(frame-name,attribute-name(variable)).

send(frame-name,selector-name(argument1,...)).

Slots specifying attribute or method have arbitrary facets which are additive knowledge to those slots (table 1).

3) Inheritance of Knowledge

Attributes, methods and other additive knowledge of a frame are inherited from super frames of that frame (see fig. 4). FLORES allows multiple inheritance, and also restriction of super frames for individual slots (table 1).

4) System defined Predicates, Frames and Methods

There are a number of basic predicates provided by FLORES: message passing predicates (send etc.), retrieving attribute values predicates (get etc.), frame definition predicate (defframe), adding/deleting/replacing predicates (put/remove/replace etc.). There also provied predicates for referring to the frame environment (fig.5, #receiver in 7).

There are two system defined frames: #CLASS, which has common knowledge of all frames, and #META-CLASS, which has common knowledge of all class-level frames (fig. 3). #CLASS has frame manipulating methods, such as 'get' method whose function is almost as same as 'get' predicate. #META-CLASS has such methods

as frame instantiation method.

3.4 Example

Fig. 5 shows frames expressed in FLORES. The slots: classification, class, instance, super, sub, are system defined slots. Each of these slots' values represent the level, class frame, instance frames, super frames, sub frames of that frame, respectively.

Plural values are usually expressed by plural clauses (fig.5, 21-22). But, in some cases (fig.5, 16), plural values are expressed by only one clause which gives one value by one each time failure occurs.

Record-type slot in #file1 (fig.5, 27) gives the same record-type of input file of #check1. This kind of information is usually expressed in FRL as if-needed demon. But in FLORES same expression (svalue facet) is used both for caluculating value dynamically and for retrieving statically defined value, because both processes are same at the point of finding a value. The role of if-needed as a demon (procedure activated when some events occure, in this case, values are referred.) is separated from the role of finding a value, and is realized as if-referred demon (table 1).

Acquire slot of #operation (fig.5, 7) is a method. It is activated, for example, when a message: 'acquire(#error1)' is sent to #check1. Then, the method is executed under the current frame environment, that is #check1, according to the knowledge inheritance conversion. To say it differently, #receiver in that method is interpreted to be the frame #check1, and acquiring rules: #check and #input will be executed. This kind of

knowledge inheritance uses automatic back-tracking and cut operation of Prolog. To keep frame environment, FLORES stores frame names and necessary information expressed as clauses, and manipulates them in stacking manner.

4. Comparison with Other Systems

One of the already proposed knowledge representation systems in Prolog is knowledge representation facilities of Prolog/KR, called world. World in Prolog/KR is created by the predicate call:

(Create-world world-name predicate-definition ...),

and referred to by the predicate call:

(with world-name predicate-call ...).

Predicates defined within a world are visible only within that world, unless predicate definition of the form:

(is-a other-world-name)

is included. This function facilities frame-like representation. To this extent, world only gives one hierarchy: is-a, while FLORES permits two types of hierarchy: class/instance and super/sub hierarchy. Prolog/KR allows to nest the worlds by using 'with' predicate multiple times. FLORES does not have corresponding functions, but it seems that such function is not needed in our prospective applications.

There are two systems which have something to do with FLORES in some restrictive aspects. Miyaji et al. are proposing an idea of current data base for checking consistency from the knowledge assimilation viewpoint. Current data base, specified by the form: 'current-db(clause)', is a set of assertions. It resembles frame structure in FLORES. The current-db, which corresponds

frame in FLORES, contains the body part of a clause as well as the head part. This contrasts to FLORES's frame, where frame governs only the head part of a clause. This difference make FLORES free from developing special purpose interpreter. Miyaji et al. had to develop such interpreter: demo predicate, which interprets the current-do forms.

The other work which connects FLORES is that of Takeuchi et al. Their aim is to realize object-oriented programming in concurrent Prolog. In their system, objects are concurrent processes, and message passing is the information transfer via common variables between such processes. So objects are predicate-call1 and predicate-call2 which are basically specified by the form: 'predicate-call1//predicate-call2'. Objects in FLORES differs in that objects exist before they are called, and so it is easier to use FLORES for modeling the task domain of an application.

5. Conclusion

This paper presents FLORES: a knowledge representation system in Prolog. This system makes task domain modeling easier than the conventional knowledge representation systems in Prolog. By developing the natural language system on FLORES, we will pursue efficiency and sufficiency of FLORES.

Acknowledgement: We thank T. Hayashi, Manager and a knowledge data base group of Software Laboratory who gave us valuable comments.

References

1) Sugiyama,K., Kameda,A., Akiyama,K., Makinouchi,A., Natural Language Understanding in an Interactive Programming System, Information Processing Society of Japan, Symposium on Nautal Language Processing Technologies, pp.35-40, 1983 (in Japanese).

2) Sugiyama,K., Akiyama,K., Kameda,A., Makinouchi,A., An Experimental Interactive Natural Language Programming System, submitted to the Institute of Electronics and Communication Engineers of Japan (in Japanese).

3) Woods,W.A., Semantics and Quantification in Natural Languge Question Answering, in Advances in Computers (M.C.Yovits ed.), Vol. 17, pp.1-87, 1978.

4) Warren,D.H.D.,Pereira,F.C.N., An Efficient Easily Apaptable System for Interpretin Natural Language Queries, American Journal of Computational Linguistics, Vol.8, No.3-4, pp.110-122, 1982.

5) Roberts,R.B., Goldstein,I.P., The FRL Manual, MIT AI Lab. Memo 409, 1977.

6) Bobrow,D.G., Stefik,M., The LOOPS Manual, Memo KB-VLSI-81-13, Xerox PARC, 1983.

7) Nakashima,H., A Knowledge Representation System: Prolog/KR, Univ. Tokyo, METR 83-5, 1983.

8) Miyaji,I., Kunifuji,S., Kitagami,H., Furukawa,K., Takeuchi,S., Yokota,Y., A Knowledge Assimilation Method for Logic Database, Report at Research Institute for Mathematical Science (RIMS) of Kyoto Univ., pp.1-21, 1983 (in Japanese).

9) Takeuchi,S., Furukawa,K., Shapiro,E.Y., Object-oriented Programming in Concurrent Prolog, Proceeding of the Logic Programming Conference 5.2, pp.1-9, 1983 (in Japanese).

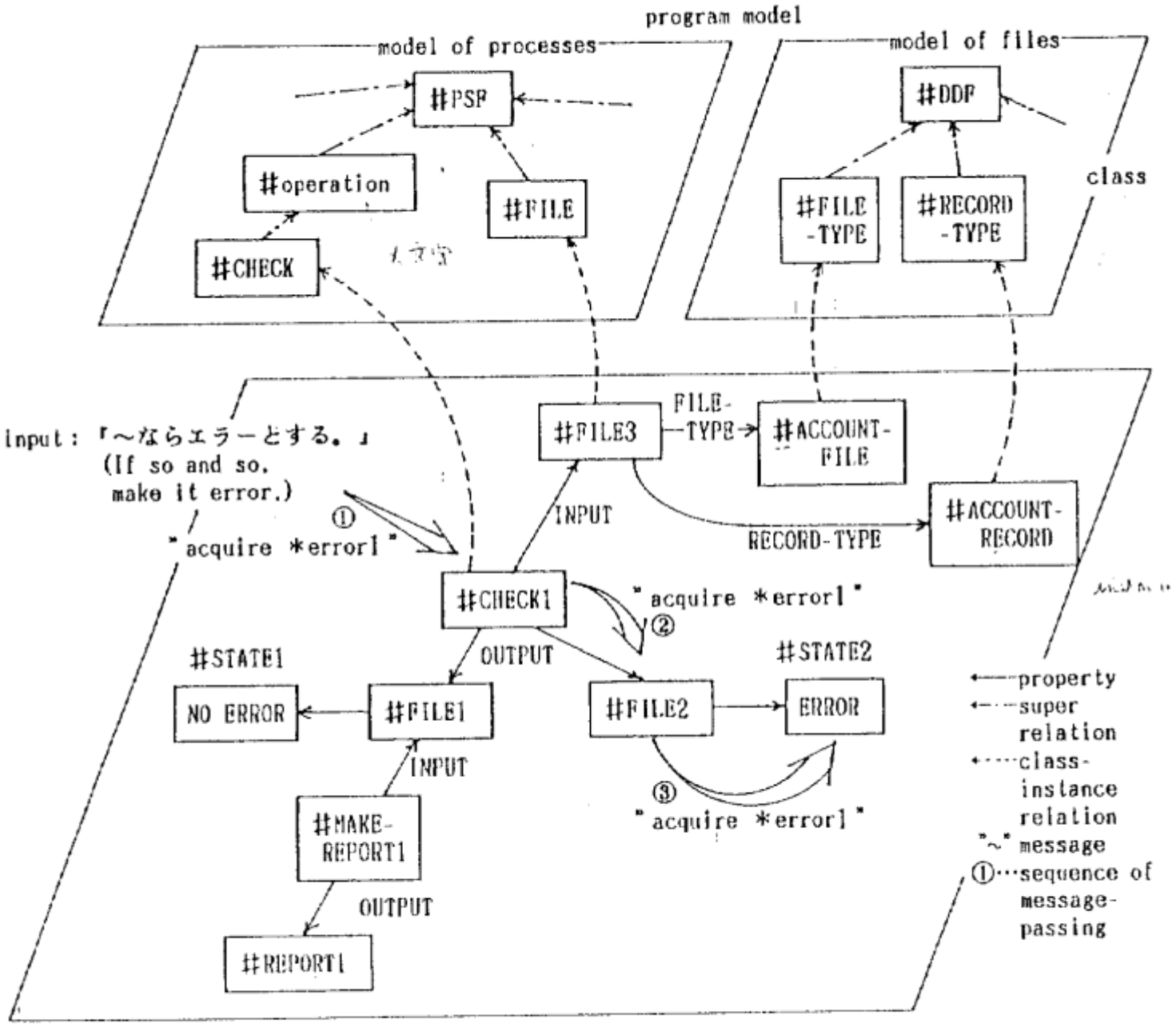

Fig. 1 Program specification acquisition

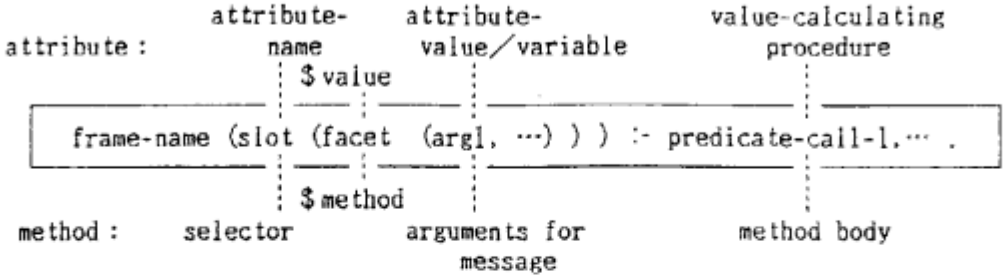

Fig. 2. Frame fragment in PROLOG

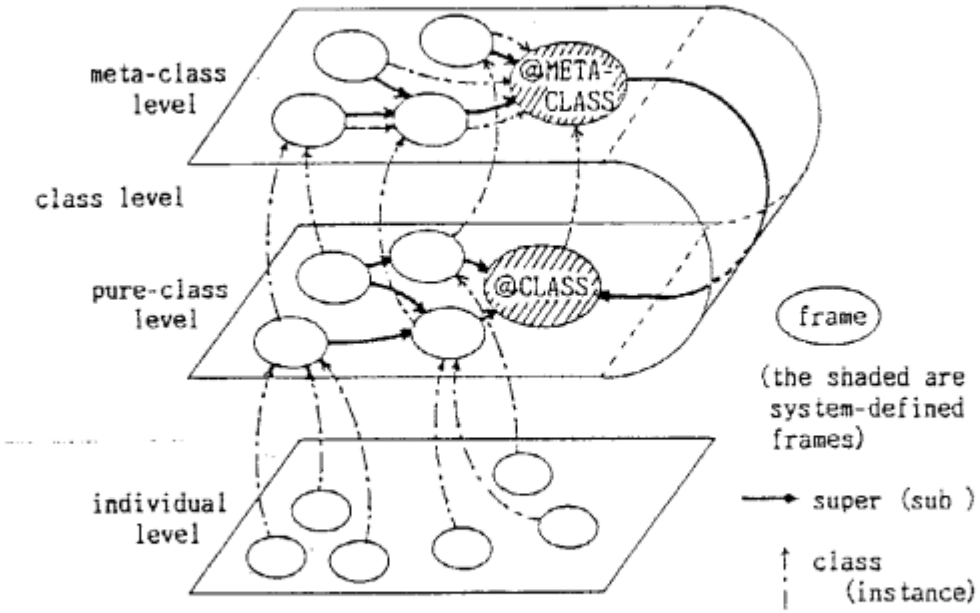

Fig. 3. Hierachy of frames

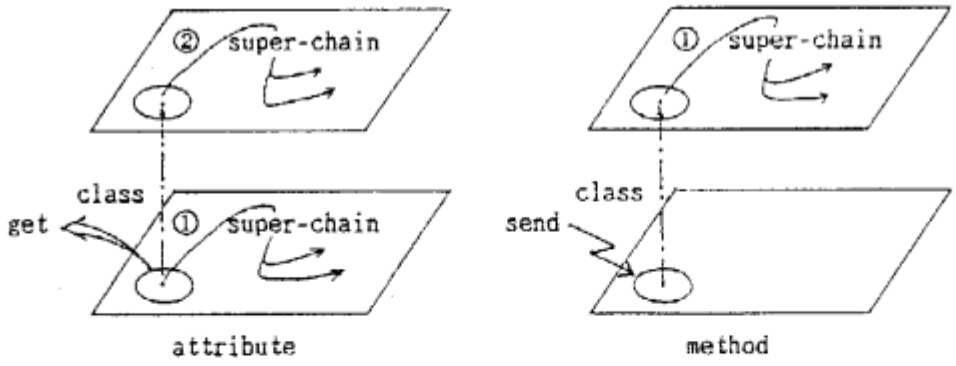

Fig. 4. Knowledge inheritance

```
①#operation  (classification ($value  (pure-class) ) ) .
②#operation  (class     ($value  (@META-CLASS) ) ) .
③#operation  (super     ($value  (#psf ) ) ) .
④#operation  (sub       ($value  (Op) :-member  (Op,  〔#check...〕 ) .
⑤#operation  (input     ($acquire  (Semantic-frame) ) )  :-...
⑥#operation  (output    ($acquire  (Semantic-frame) ) )  :-...
⑦#operation  (acquire  ($method   (Semantic-frame) ) )  :- @receiver (Name) ,
         get  (Name,acq-rule  (R ) ) ,send  (Name,R (Semantic-frame) ) , ! ,
         set-of (_,  ( get@ (Name,R ($exec ( (Slot,Facet) ) ) ) ,
                       send@ (Name, Slot (Facet  (Semantic-frame) ) ) ) . _) .
⑧#operation  (*input  ($method (Semantic-frame) ) )  :- ...
⑨#operation  (*input  ($exec ( 〔input, $acquire 〕 ) ) ) .
                      .
                      .

⑩#check  (classification ($value  (pure-class) ) ) .
⑪#check  (class     ($value  (@META-CLASS) ) ) .
⑫#check  (super     ($value  (#operation ) ) ) .
⑬#check  (sub       ($value  (#check1) ) ) .
⑭#check  (output    ($condition  (Semantic-frame) ) )  :-...
⑮#check  (acq-rule ($value  (Rule) ) )  :-member  (Rule,  〔*check, *input 〕 ) .
⑯#check  (*check  ($method (Semantic-frame) ) )  :-...
⑰#check  (*check  ($exec (Subgoal ) ) )  :-member  (Subgoal,
             〔 〔input, $acquire 〕 ,  〔output,  $acquire 〕 ,
               〔output,  $condition 〕 〕 ) .
                      .
                      .

⑱#check1 (classification ($value  (individual) ) ) .
⑲#check1 (class     ($value  (#check ) ) ) .
⑳#check1 (input     ($value  (#file3 ) ) ) .
㉑#check1 (output    ($value  (#file1 ) ) ) .
㉒#check1 (output    ($value  (#file2 ) ) ) .
                                          ↖
                                  send (#check1,acquire (*error1) )

㉓#file1  (classification ($value  (individual) ) ) .
㉔#file1  (class     ($value  (#file) ) ) .
㉕#file1  (output-of  ($value  (#check1) ) ) .
㉖#file1  (input-of ($value  (#make-report1) ) ) .
㉗#file1  (record-type  ($value  (Rec ) ) )  :- @receiver (Name) ,
      get (Name,output-of (Op) ) ,get (Op,input (File) ,
      get (File,record-type (Rec ) ) ) .
```

Fig. 5. Frame representation by PROLOG clauses

Table 1. System defined additive knowledge

category	facet	argument	meaning
attribute /method	$super	super frames	restriction of super frames
attribute	$default	attribute-value	default value
	$constraint		constraint
	$if-referred		demon activated when value is retrieved
	$if-added		demon activated when value is added
	$if-removed		demon activated when value is deleted
method	$if-called	argument for message	demon activated when method is executed