A dedicated language for distributed intelligence based fuzzy sensors

Eric Benoit, Eric Chotin, Gilles Mauris

A Dedicated Language for Distributed Intelligence-Based Fuzzy Sensors

E. Benoit, E. Chotin and G. Mauris
LAMII/CESALP, Université de Savoie
41 Avenue de la Plaine, BP 806,74016 Annecy, France
Ph : +33 450 66 60 44 Fax : +33 450 66 60 63 E-Mail : benoit@univ-savoie.fr WWW: http: //ava.univ-savoie.fr/

Abstract **This paper presents a concept of distribution of the computational activity over a networked set of fuzzy sensors. This concept is based on the separation of the concept of intelligence and the computational capability. The PLICAS language specially created to apply this concept and its fuzzy processing capabilities are presented. This concept is applied to the fuzzy description of comfort measurement from temperature and humidity measurements.**

I. INTRODUCTION

Since the eighties, the concept of smart cells communicating over a field bus network in order to drive an industrial process has been developed [1][2]. It is commonly admitted that smart cells (i.e. smart sensors or smart actuators) include functionalities such as measurement, communication, configuration and validation [3]. These functionalities are more and more complex. They now include diagnosis, auto-configuration, fuzzy processing, decision making. Usually, smart sensors include a processing unit whose computing power is able to perform these software functions. From our point of view, we consider that the definition of intelligence is independent of the computing power.

In the proposed approach, smart sensors own the definition of the software functionalities but are not always able to execute them locally. Thanks to the network, these software functionalities are sent to a smart sensor, to a smart actuator or to a common resource that has higher computation facilities. Due to the wide range of possible processing units, the exchange of software functionalities takes place at the source level. A dedicated language, called PLICAS, has been especially developed to design smart sensors integrating fuzzy functionalities [4][5]. In order to perform the remote execution of the PLICAS code, each unit owns a PLICAS compiler with computing capabilities. The presence of this compiler insures the interchangeability of smart cells and allows a good base for interoperability.

II. DISTRIBUTION OF COMPUTATIONAL ACTIVITY

The way we choose to improve the distribution of computational activity is to implement a compiler into each smart cell, i.e. sensor or actuator. Each smart cell possesses a general behavioural description which is a state machine describing the different functionalities to be executed by the cell according to the events which occur. In our approach, the general behavioural description is coded with a PLICAS language source code.

Some smart cells do not have enough computational capabilities to perform their intelligent functionalities, for example if the cell has a small processor or no processor at all. In this case, our approach is to let another cell compile the source code of the small cell. The smart cell with a small processor has its PLICAS source code and sends it over the network at the initialization. Then, a server cell gets the PLICAS code, performs the compilation and executes it.

For example, imagine a car with an indicator which performs auto-diagnostic in order to know if its light is in good or not. If not it can then use the stop light instead of its own light. The solution we propose for making that, is to use a processorless indicator with a network communication interface and a read only memory which contains the definition of these functionnalities. The indicator can then send definitions to an other system with processor, like the car radio, which performs them.

In our approach, we consider that the intelligence of a smart cell is defined by its general behavioural description an not by its computational capacity. This firstly implies that each smart cell owns its general

behavioural description in a source code, secondly that at least one cell on the network provides computational service.

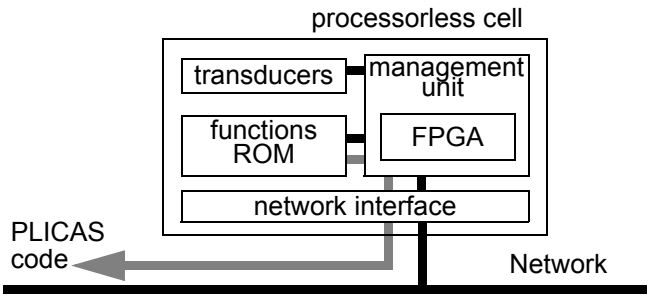

Fig. 1. Example of smart cell without processor

III. FUZZY SENSORS

A. Fundamentals

Fuzzy symbolic sensors are based on the translation of information from a numerical representation to a symbolic one. To perform a symbolic measurement, it is necessary to clearly specify the relation between symbols and numbers. Let X be the universe of discourse associated with the measurement of a particular physical quantity. Denote x any element of X. In order to symbolically characterize any measurement over X, let L be a set of words, representative of the physical phenomenon. For example, the set L={***cold***, ***mild***, ***hot***} could be used to represent a temperature. Denote $F(E)$ the set of the fuzzy subsets of a set E.

Introduce an injective mapping M : $L \rightarrow F(X)$, called the *fuzzy meaning* of a symbol (see Zadeh [6]). It associates any symbol L of L with a fuzzy subset of X. The fuzzy meaning of a symbol L is characterized, for all $x \in X$, by its membership function denoted $\mu_{M(L)}(x)$.

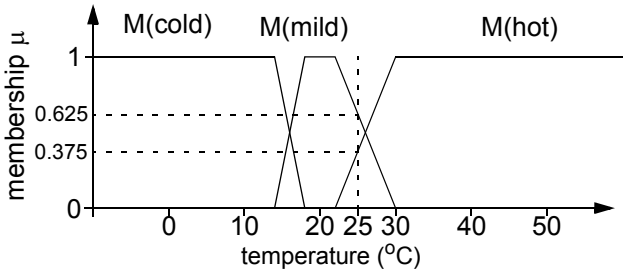

Fig. 2. Fuzzy meanings of symbols cold, mild and hot

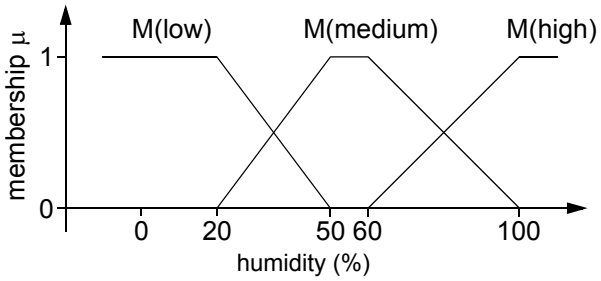

Fig. 3. Fuzzy meanings of symbols low, medium and high

Another mapping D : $X \rightarrow F(L)$, called the *fuzzy description* of a measurement over L associates any measurement of X with a fuzzy subset of symbols of L. The fuzzy description of a measurement is characterized, for all $L \in L(X)$, by its membership function $\mu_{D(x)}(L)$. Any measurement that belongs to the meaning of a symbol, can obviously be symbolically described at least by this symbol. Therefore, the description of a measurement is linked to the meaning of a symbol by the following relation:

$$\mu_{D(x)}(L) = \mu_{M(L)}(x) \qquad (1)$$

It means that if a symbol belongs to the description of a measurement at a grade of membership $\mu_{D(x)}(L)$, then the measurement belongs to the meaning of the symbol at the same grade of membership. Let X = [-10, 60] be the universe of discourse for the measurement of temperatures expressed in Celsius. The fuzzy meanings of ***cold***, ***mild***, and ***hot*** are represented by the membership functions (plotted Fig. 2.). The fuzzy description of the measurement x = 25 °C is obtained according to Eq. (1).

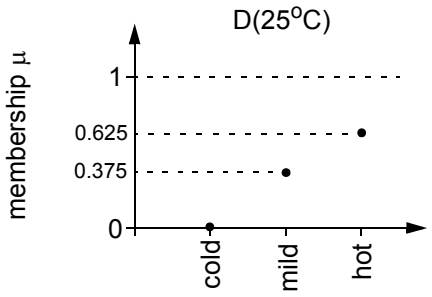

Fig. 4. Fuzzy description of 25 °C

In order to make easier the construction of a partition , a fuzzy meaning can be defined relatively to another

one by the use of linguistic modifiers. For example the meaning of ***cold*** can be defined relatively to the meaning of ***mild*** by the use of the linguistic modifier "under" which moves down the considered meaning. And ***hot*** can be defined relatively to the meaning of ***mild*** by the use of the linguistic modifier "over" which moves up the considered meaning.

M(cold) = under(M(mild)) (2)

M(hot) = over(M(mild)) (3)

The aggregation of two measurements can be performed with a set of rules [7]; such as: "if the temperature is ***hot*** and the humidity is ***low*** then ambiant is ***uncomfortable***". The set of all rules needed for an aggregation can be represented by a table as follow.

TABLE I

TABLE USED FOR THE EXAMPLE OF COMFORT AGGREGATION

T \ H	low	medium	high
hot	uncomfortable	uncomfortable	uncomfortable
mild	uncomfortable	comfortable	uncomfortable
cold	uncomfortable	uncomfortable	uncomfortable

IV. THE PLICAS LANGUAGE

PLICAS is a language created to manage the sequencing of predefined functions. It includes basic arithmetic and logical functions needed to perform this sequencing. Specialized functions like FFT or rule based controller can be added to the language during a technological configuration. Added functions are then used as primitive functions of the language. In order to minimize the size of the compiler code, the language uses only global variables and does not allow new function definitions. In its present version the compiler is 32Kbytes large and can be implemented with small processors.

A PLICAS source code is partitioned into 3 sections which are the *declaration block*, the *initialisation block* and the *main bloc*. In the *declarations* bloc, types of all variables used in the program are defined. Specific types for the management of fuzzy variables and the management of networked variables are used. At the beginning of the execution of a PLICAS code, the initialisation block is executed one time. Then the main block is executed periodically.

A. Fuzzy specific types and instructions

The variable type *partition* is a type of variable which contains the universe of discourse and a set of symbols and their fuzzy meanings. If the partition is the aggregation of 2 other partitions, it contains only a set of symbols.

A variable of type *fuzzy* is a fuzzy description, i.e. a set of symbols and their membership degrees. Before it is used, it has to be associated with a partition. Indeed, fuzzy meanings of symbols are included in a partition.

The aim of the function def_partition() is to initialize a partition with a symbol and its fuzzy meaning and the universe of discourse. The fuzzy meaning is characterized by 4 parameters. The universe of discourse is characterized by its lower and upper bounds.

For example the instruction : def_partition(humidity,"medium",20,50,60,100,10,100) initializes the partition *humidity* with the meaning of the symbol "medium" and the universe of discourse [10,100]. Meanings of other symbols are defined with the functions under() and over().

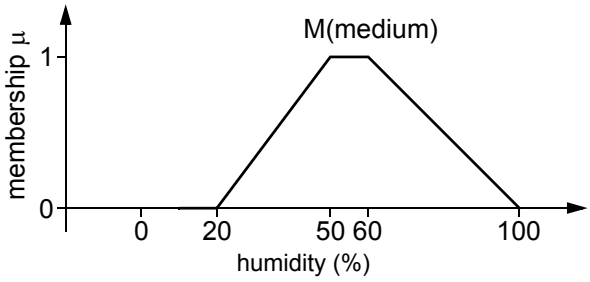

Fig. 5. Meaning of the symbol "medium"

A function *fuzz()* perform a fuzzy description of a numeric variable. For example if T is a fuzzy description (type *fuzzy*) , then T=fuzz(25) compute the result shown in Fig. 4. T is then the fuzzy set {0/cold, 0.375/mild, 0.625/hot}.

B. Network specific types and instructions

For our applications, we use a network which supports communication by diffusion. In this kind of communication, no recipient information is needed. Each data packet is identified by a number which describes the content of the packet. All smart cells listen the network, and choose to get the packet or not depending on the identifier. In our applications, we choose to associate each variable with an identifier. This identifier is chosen by the compiler.

For variables which have to be imported from or exported to the network, PLICAS proposes 2 modifier types: *extern* and *public*. The *extern* modifier indicates to the compiler that the declared variable will be received through the network. The *public* modifier indicates to the compiler the declared variable will be sent to the network.

The reception of an extern variable is performed by the function *import(variable).* If this variable has not been received since the beginning of the current cycle, the function waits for the variable.

The emission of a public variable is performed when this variable is affected. For example if y is a public variable, then the instruction "y = 4" sends the variable y with the value 4 on the network.

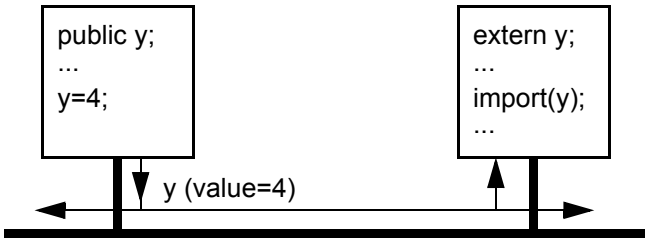

Fig. 6. Management of network variables

V. APPLICATION TO THE AGGREGATION OF INFORMATION TROUGH THE NETWORK

A. Presentation

The application chosen to illustrate the concept of a weak cell is a description of comfort by aggregation by means of a set of rules of temperature and humidity measurements. These values are sent through the network to a computer which has a PLICAS compiler. The program needed to compute the fuzzy description of humidity and temperature and also to aggregate by means of a set of rules, has been sent through the network by the sensor. The resulting fuzzy description of comfort is then put back on the network in order to be used by an actuator or other computers.

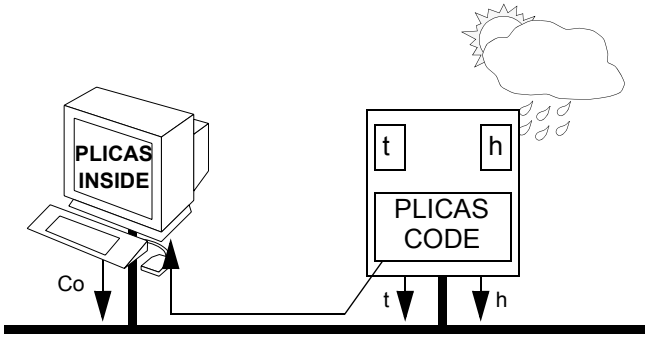

Fig. 7. Schema of the application

B. Main program

In the main program, we have the declarations of the network variables: sync for the synchronization by another cell or process dedicated to time management and h and t which are produced by the smart sensor. We have also the three partitions and three variables H,T,C for the handling of fuzzy descriptions of temperature, humidity and comfort.

The main block sequences the general behaviour. The result shown on the computer screen is for example:

H= 0.2/low, 0.8/medium
T= 0.4/mild, 0.6/hot
C= 0.32/comfortable, 0.68/uncomfortable

Here is the main code:

```
declarations
        extern double sync;
        extern double h,t;
        public fuzzy Co;
        double cpt;
        partition humidity,temperature,comfort;
        fuzzy H,T,C;
bloc initialisation
...
bloc inference
...
bloc main
        import(sync);
```

```
import(h);
H=fuzz(h);
import(t);
T=fuzz(t);
show(H);
show(T);
execute(inference);
show(C);
Co=C;
if sync !=0 then stop(0);
```

C. Initialization bloc

In the initialization bloc, we find the partition definitions

```
bloc initialization
def_partition(humidity,"medium",20,50,60,100,
        10,100);
under(humidity,"medium","low");
over(humidity,"medium","high");

def_partition(temperature,"mild",14,18,22,30,10,60);
over(temperature,"mild","hot");
under(temperature,"mild","cold");

def_termes(comfort,2,"comfortable",
        "uncomfortable");
H in humidity;
T in temperature;
C in comfort;
Co in comfort;
cpt=0;
```

D. Inference bloc

The PLICAS language is specially designed to allow a simple coding and inferencing of a set of rules:

```
bloc inference

if T is "hot" and H is "low"
        then C is "uncomfortable";
if T is "hot" and H is "medium"
        then C is "uncomfortable";
if T is "hot" and H is "high"
        then C is "uncomfortable";
if T is "cold" and H is "low"
        then C is "uncomfortable";
if T is "cold" and H is "medium"
        then C is "uncomfortable";
if T is "cold" and H is "high"
        then C is "uncomfortable";
if T is "mild" and H is "low"
        then C is "uncomfortable";
if H is "mild" and T is "mild"
        then C is "comfortable";
if T is "mild" and H is "high"
        then C is "uncomfortable";
```

This set of rules is inferred according to Zadeh's compositional rule of inference using for the fuzzy intersection operator the product, and for the fuzzy union operator the bounded sum. If desired, other fuzzy operators can be selected by the user.

VI. CONCLUSION

In this paper, we have shown that it is possible to implement complex functionnalities in a smart sensor or smart actuator even if its computational capacity is too weak for these functionnalities. This concept can be an interesting solution for the implementation of smart functionnalities in processorless sensors. The use of a unique language for the description of the functionnalities of each smart cell improves the interchangeability.

REFERENCES

[1] Ren C. Luo, M.H. Lin, R.S. Scherp, *"Dynamic multisensor data fusion system for intelligent robots"*, IEEE Journal of robotics and automation, Vol. 4, No 4, Aug. 1988, pp. 386-396.

[2] Yagsu T.,*"Support system to construct distributed communication networks - Implementation"*, Proc. of the 1st European Congress on Fuzzy and Intelligent Technologies (EUFIT 93), Aachen, Germany, Sept. 93, pp. 532-536.

[3] M. Staroswiecki, M. Bayart "*Models and Languages for the Interoperability of Smart Instruments*", Proc. of the 2nd IFAC Symposium on Intelligent Components and Instruments for Control Applications, SICICA 94, Budapest, Hungary, june 94, pp 1-12.

[4] Josserand J.F. and Foulloy L., "Fuzzy components network for intelligent measurement and control", IEEE trans. on Fuzzy Systems, Vol 4, No 4, Nov 1996, pp 476-487.

[5] Benoit E., Chotin E., Foulloy L., "Processorless smart sensors with distributed intelligence", XIV IMEKO world congress, New measurements chalenges and visions, Tampere-Finland, June 1997,Vol. 5, pp.60-65.

[6] Zadeh L.A., "Quantitative fuzzy semantics", Information Sciences, Vol. 3, 1971, pp. 159-176.

[7] Mauris G., Benoit, E. and Foulloy L., "The aggregation of complementary information via fuzzy sensors" , Measurement, Vol 17, No 4, 1996, pp. 235-249.