International Journal of Computational Intelligence Research.
ISSN 0973-1873 Vol.1, No.2 (2005), pp. 171–182

Using FML and Fuzzy Technology in Adaptive Ambient Intelligence Environments

Giovanni Acampora and Vincenzo Loia

Dipartimento di Matematica e Informatica
Universitá degli Studi di Salerno
Via Ponte don Melillo, 84084 Fisciano, Salerno, Italy
{*gacampora, loia*}*@unisa.it*

Abstract: **Ambient Intelligence (AmI, shortly) gathers best results from three key technologies, Ubiquitous Computing, Ubiquitous Communication, and Intelligent User Friendly Interfaces. The functional and spatial distribution of tasks is a natural thrust to employ multi-agent paradigm to design and implement AmI environments. Two critical issues, common in most of applications, are (1) how to detect in a general and efficient way context from sensors and (2) how to process contextual information in order to improve the functionality of services. In this work we experiment a framework where hybrid techniques (distributed fuzzy control, mobile agents, fuzzy rules induction algorithms) are mixed to gain flexibility and uniformity.**
Keywords: Distributed Fuzzy Control, Markup Languages, Agents, Fuzzy Rule Induction Algorithms.

I. Introduction

When designing AmI Environments [Aarts2004], different methodologies and techniques have to be used, ranging from materials science, business models, network architectures, up to human interaction design. However, as key technologies, AmI is characterizes by:

- *Embedded.* Devices are (wired or unwired) plugged into the network [Ditze2004]. The resulting system consists of several and multiple devices, compute equipments and software systems that must interact among them. Some of the devices are simple sensors, other ones are actuator owning a crunch of control activity on the environment (centralheating, security systems, lightning system, washing machines, refrigerator, etc.). The strong heterogeneity makes difficult a uniformed policy-based management.

- *Context aware.* This term appeared for the first time in [Schilit94], where the authors defined context as location, identities of nearby people and objects, and changes to those objects. Many research groups have been investigating on context-aware applications, but there is no common understanding what context and context awareness exactly means. Roughly, the system should own a certain ability to recognize people and the situational context.

- *Personalized.* AmI environments are designed for people, not generic users. This means that the system should be so flexible to tailor itself to meet human needs.

- *Adaptive.* The system, being sensible to the user's feedback, is capable to modify the corresponding actions have been or will be performed[Astrom1987].

We have designed and implemented an intelligent home environment populated by intelligent appliance agents skilled to perform distributed and adaptive transparent fuzzy control. The agents interact and coordinate their activities using the Fuzzy Markup Language (FML) [Loia2005] as abstract protocol over shared resources, independently from hardware constraints. The agents compose an abstract layer that binds the instrumental scenario with the services, assuring efficiency and adaptivity. This approach allows AmI designers to achieve useful goals:

- to customize the control strategy on specific hardware through an automatic procedure;

- to distribute fuzzy control flow in order to minimize global deduction time and better exploit the natural distributed knowledge repositories;

- to acquire, at run time, the user's behavior and environment status in order to apply context-aware adaptivity.

In the remainder of this paper, we will describe our solution that attempts to solve these complicated problems. First, we will describe how we have used XML-derived technologies

in order to define FML, a markup language skilled for defining detailed structure of fuzzy control independent from its legacy representation. Then, we will show the gain achievable in terms of distribution and concurrent execution by means of agent technology. As last issue, we will describe an adaptive methodology, based on inductive computation, that allows fuzzy rules to modify themselves according to user's behaviors.

II. Fuzzy Markup Language

Initially, FML has been designed to act like a middleware between the Legacy Fuzzy Environment and the real implementation platform. Legacy Fuzzy Environment module allows to create a fuzzy controller using a legacy representation [Acampora2005]. An example of Legacy Fuzzy Environment module is Matlab^{TM}that produce a *.fis* file to represent the fuzzy control system. The obtained legacy fuzzy controller is passed to the FML Converter module that translates it into a markup-based description (FML language). Next step concerns the real implementation of the fuzzy controller on specific hardware. Initial version of FML used XSLT [XSLT2001] languages translator to convert FML fuzzy controller in a general purpose computer language using an XSL [XSL2003] file containing the translation description. At this level, the control is compilable and executable for proposed hardware.
Actually, FML can be considered a standalone language used to model the fuzzy controllers from scratch. Now, the FML compiler is based on the integration of different techonologies able to istantiate a runtime fuzzy controller without additional work. These technologies are : the *TCP/IP Client/Server application* and the *JAXB XML Binding technology*. In particular, the JAXB XML Binding technology allows to generate a Java classes hierarchy starting from the FML control description. The TCP/IP Client/Server application allows to separate the real control from the controlled devices in order to obtain the total independence of the devices from the language used to code the fuzzy controller. In particular, a TCP Server instantiates a Java objects collection representing the FML controller starting from the classes hierarchy generated by JAXB module. Using this object collection, the Server will be able to apply the inference operators on the objects representing the fuzzy rule base, generating, in this way, a set of defuzzificated values representing the control results.
The TCP Client, hosted on the controlled devices, is a simple standard TCP Client able to send the sensors value to Server and to receive the control results; from this point of view, the Client does not know the details about the fuzzy control, it sees only a bidirectional data flow. The Client/Server communication is performed, obviously, by TCP Sockets. Figure 1 shows the architecture of proposed system. This choice allows to obtain an high level of abstraction showing an only gap: the client and the server have to be agree on the exchange data format. In particular, the server have to know, exactly, the data format coming from the client and vice versa. The proposed systems uses the classic string data type to solve the problem. In particular, in order to exchange the sensors values and the inferred results, the client and server have to choose a special character to create a communication data string. While the client uses this character to compose a string containing the sensors data, the server uses the same character to infer and to create a string containing the fuzzy control results. Client and Server, simply, have to split the received string in order to use the data in a normal fashion. Figure 2 shows the communication step performed during a control iteration by the system.
In order to accomplish the JAXB/TCP/FML Controller is necessary to create a TCP *endpoint* able to identify in a direct and unambiguous way the FML Server on the Internet. TCP defines an endpoint to be a pair of integers $(host, port)$, where the *host* is the IP address for the FML Server host and *port* is a TCP port where the server is executed on that host. The IP address depends from the network which hosts the FML Server; the TCP port has to be defined in univocal way to allows the FML clients to contact the Server without problems. The FML Server port is defined considering the concatenation of ASCII codes related to **F**, **M** and **L** characters modulo 65536(available TCP ports) obtaining, in this way, the integer port number 12330. Some examples of FML/TCP endpoint are: $(192.168.0.4, 12330)$, $(193.205.186.85, 12330)$.

A. Fuzzy Markup Language(FML) and Fuzzy Logic Control(FLC)

Since Zadeh's coining of the term fuzzy logic [Zadeh65] and Mamdani's early demonstration of Fuzzy Logic Control (FLC) [Mamdani74], an enormous progress has been done by the scientific community in the theoretical as well as application fields of FLC. Trivially, a fuzzy control allows the designer to specify the control in terms of sentences rather than equations by replacing a conventional controller, say, a *PID* (proportionalintegral- derivative) controller with linguistic IF-THEN rules. The main components of a fuzzy controller are:

- Fuzzy Knowledge Base
- Fuzzy Rule Base
- Inference Engine
- Fuzzification sub-system
- Defuzzification sub-system

The Fuzzy Knowledge Base contains the knowledge used by human experts. The Fuzzy Rule Base represents the set of relations among fuzzy variable defined in the controller system. The Inference Engine is the fuzzy controller component able to extract new knowledge from fuzzy knowledge base

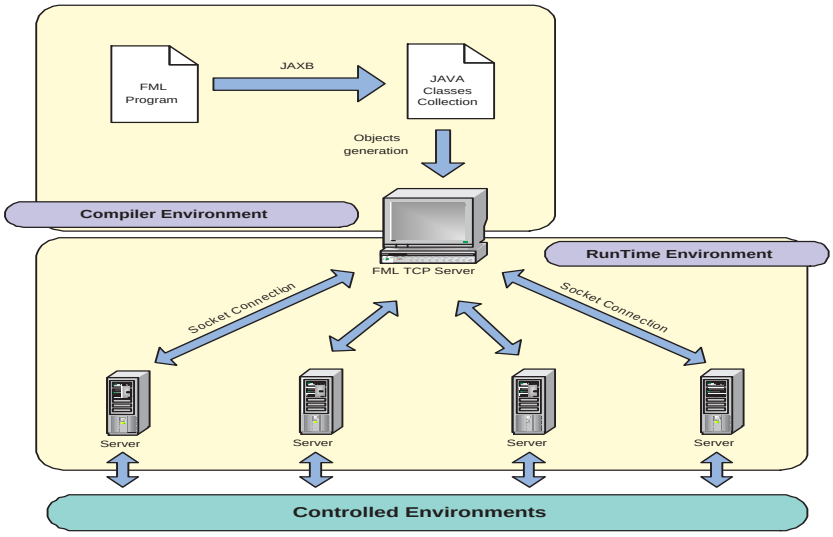

Figure. 1: FML TCP/JAXB Architecture

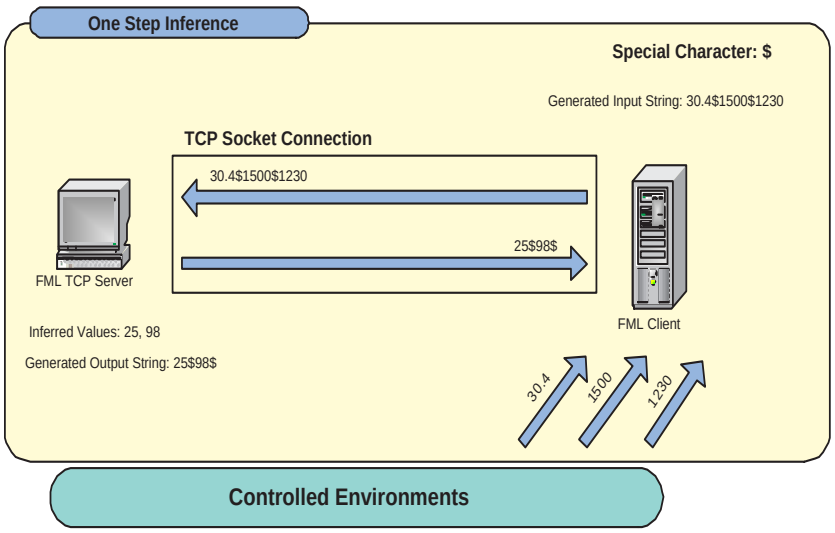

Figure. 2: FML TCP/JAXB Inference Step

and fuzzy rule base. Extensible Markup Language (XML) [DuCharme99] is a simple, very flexible text format derived from SGML (ISO 8879). Originally designed to meet the challenges of large-scale electronic publishing, nowadays XML plays a fundamental role in the exchange of a wide variety of data on the Web, allowing designers to create their own customized tags, enabling the definition, transmission, validation, and interpretation of data between applications, devices and organizations. If we use XML, we take control and responsibility for our information, instead of abdicating such control to product vendors. This is the motivation under FML proposal: *to free control strategy from the device.* FML uses:

- XML in order to create a new markup language for FLC;
- XML Schema in order to define the legal building blocks of an XML document;
- XSLT in order to convert fuzzy controller description into a programming language code.

Initially, FML used the XML Document Type Definition (DTD) to realize the context free grammar for the new markup language. Actually, the FML grammar is defined by the XML Schema in order to allows a direct integration with the JAXB techonology used in the FML compiling step. Using XML Schema and JAXB is possible to map a detailed logical structure of a fuzzy controller basic concepts of FLC into a tree structure, as shown in figure 3, where each node can be modeled as a FML tag, and the link father-child represents a nested relation between related tags. This logical structure is called *FOM*(Fuzzy Object Model).
Currently, we are using FML for modeling two well-known fuzzy controllers: Mamdani and Takagi- Sugeno-Kang (TSK) [Takagi85].

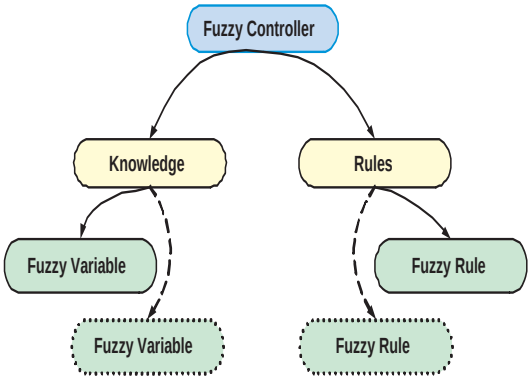

Figure. 3: Fuzzy Control Tree

In order to model the Controller node of fuzzy tree, the FML tag <**FUZZYCONTROL**> is created (this tag opens any FML program). <**FUZZYCONTROL**> uses three tags: *type*, *defuzzifyMethod* and *ip*. The type attribute permits to specify the kind of fuzzy controller, in our case Mamdani or TSK; defuzzifyMethod attribute defines the defuzzification method; ip can be used to define the location of controller in the computer network and, in the case of <**FUZZYCONTROL**> tag it defines the first member of TCP endpoint pair. Considering the left sub-tree, the knowledge base component is encountered. The fuzzy knowledge base is defined by means of the tag <**KNOWLEDGEBASE**> which maintains the set of fuzzy concepts used to model the fuzzy control system. <**FUZZYVARIABLE**> defines the fuzzy concept, for example Luminosity; <**FUZZYTERM**> defines a linguistic term describing the fuzzy concept, for example low; the set of tags defining the shapes of fuzzy sets are related to fuzzy terms. The attributes of <**FUZZYVARIABLE**> tags are: *name*, *scale*, *domainLeft*, *domainRight*, *type*, *ip*. The *name* attribute defines the name of fuzzy concept (i.e. time of the day); *scale* defines how to to measure the fuzzy concept (i.e. hour); *domainLeft* and *domainRight* model the universe of discourse of fuzzy concept in terms of real values (i.e. [0000, 2400]); the role of variable (i.e. independent or dependent variable) is defined by *type* attribute; *ip* locates the position of fuzzy knowledge base in the computer network. <**RULEBASE**> permits to build the rule base associated with the fuzzy controller. The other tags related to this definiton are: <**RULE**>, <**ANTECEDENT**>, <**CONSEQUENT**>, <**CLAUSEA**>, <**CLAUSEC**>, <**VARIABLE**>, <**TERM**>, <**TSKPARAM**>. The meaning of these tags appears evident and we do not further detail here.

B. Distributed Fuzzy Control

Just to give a concrete example, considering automatic lighting system, we can model the knowledge base and rule base FML code portion, and in particular the lamp light level as shown in listing 1 (Mamdani method).

```
<!DOCTYPE FUZZYCONTROL SYSTEM "fml.dtd">
<FUZZYCONTROL defuzzifymethod = "CENTROID"
    ip = "localhost" type = "MAMDANI">
<KNOWLEDGEBASE IP = "localhost">
<FUZZYVARIABLE
    domainleft = "0" domainright = "1"
    ip = "localhost" name = "Luminosity"
    scale = "Lux" type = "INPUT">
    <FUZZYTERM name="low">
        <PISHAPE
            param1 = "0.0"
            param2 = "0.45">
        </PISHAPE>
    </FUZZYTERM>
    <FUZZYTERM name="MEDIUM">
        <PISHAPE
            param1 = "0.49999999999999994"
            param2 = "0.44999999999999996">
```

```
            </PISHAPE>
        </FUZZYTERM>

        <FUZZYTERM name="HIGH">
            <PISHAPE
                param1 = "0.5501"
                param2 = "1">
            </PISHAPE>
        </FUZZYTERM>
    </FUZZYVARIABLE>

    </KNOWLEDGEBASE>

    <RULEBASE
        inferenceengine = "MINMAXMINMAMDANI"
        ip = "localhost">
        <RULE connector = "AND" ip = "localhost"
            weight = "1">
            <ANTECEDENT>
                <CLAUSE not = "FALSE">
                    <VARIABLE> Luminosity </VARIABLE>
                    <TERM> low </TERM>
                </CLAUSE>
                <CLAUSE not = "FALSE">
                    <VARIABLE> hour </VARIABLE>
                    <TERM> morning </TERM>
                </CLAUSE>
            </ANTECEDENT>
            <CONSEQUENT>
                <CLAUSE not = "FALSE">
                    <VARIABLE>dimmer</VARIABLE>
                    <TERM>medium</TERM>
                </CLAUSE>
            </CONSEQUENT>
        </RULE>
    ...
    </RULEBASE>
    </FUZZYCONTROL>
```

Listing 1: FML sample program

This crunch of FML code is useful to understand how is possible to associate a fuzzy control activity (knowledge base and eventually the rule base) on a single host (in our example localhost). In this naïve example, a centralized Mamdani fuzzy controller is produced, but in real cases, a distributed approach is performed, as illustrated in figure 4. This feature is useful to obtain several advantages:

1. to parallelize the fuzzy inference engine reducing inference time and minimizing knowledge base and rule base occupancy;

2. to manage distributed knowledge environment, i.e. environments in which the global knowledge is shared on many points of interested environment, as often happens in AmI;

3. to exploit mobile agents as a natural and efficient technology to share data distribution and dispatch running code on a network. This last concept is deepened in the last part of the paper.

In order to distribute fuzzy controller components on different hosts we need to characterize the independent members of controller. In particular, working with Mamdani we identify the following components:

- Fuzzy Controller
- Knowledge Base
- Fuzzy Variable
- RuleBase
- Fuzzy Rule

Default value of ip attribute of <**FUZZYCONTROL**> tag is localhost. The internet address value of fuzzy controller is distributed towards the bottom in the parse tree related to fuzzy program. From this point of view, the internet address of other independent components (knowledge base and rule base), if not defined, is overlapped by network address from <**FUZZYCONTROL**> tag. This distributive concept is also extended to the nodes of the parse tree related to the rule base and knowledge base: each member of the controller is spread in a scalable and distributed way, as shown in figure 4. Comparing figures 3 and 4 we better note the strong differences between a centralized controller and a distributed one. In Figure 3, all components of centralized controller are connected by straight lines indicating that all components (knowledge base, rule base and related sub components) are maintained on the same host at the same time. Figure 4 shows a distributed fuzzy controller, the whose members, connected by dotted lines, can be treated concurrently by different processes. In particular, Figure 4 shows a distributed fuzzy controller with Luminosity and Time of the day concepts hosted on 192.168.0.4, Dimmer concept hosted on 192.168.0.8 and the rule base shared on 192.168.0.5, 192.168.0.6, 192.168.0.7. In this way, we can distribute fuzzy rule base in the network and exploit distributed processing by minimizing inference. In order to address in a high-level way the issues of delocalization and concurrency, we map the distributed fuzzy model coded in FML on a multi-agents system. In particular, the agents that compose the system are: Stationary Fuzzy Agent set, Registry Agent, Inference Agent. The set of Stationary Fuzzy Agent is used to manipulate in a distributed way the concept coded in FML program and modeled in the distributed fuzzy controller. These agents represent the run time containers able to execute the fuzzy logic operator on modeled entity. Stationary Fuzzy Agents are hosted on different host of network; these hosts represent the Fuzzy Control Network (FCN). The Inference Agent is a migrating agent able to apply the classic inference operator, like *Mamdani MinMaxMin* or *Larson Product* on hosts running the Stationary Agents. Due to the delocalization of rules, it is necessary to collect

the partial inference results. This is done by the migration of Inference Agent that gathers the partial results available on each Stationary. Just to give an idea of the gain from shifting from a centralized to a decentralized evaluation, we give in figure 5 the results from a testbed done by spreading the control over three computational nodes. On the axis x we report the number of fuzzy rules evaluated, the required inference time, expressed in milliseconds, is on axis y.

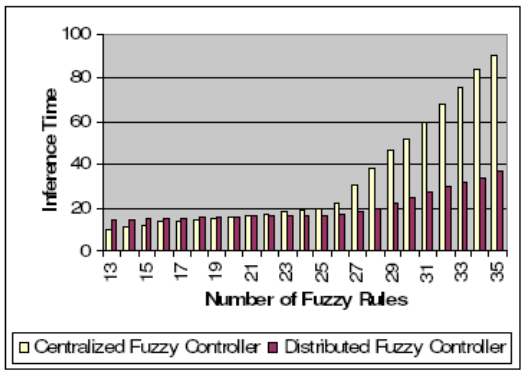

Figure. 5: Multi-Agent System Performance

III. Context-Aware Adaptivity

Such layer implements an evolutionary-based mechanism [Yuhui99] in order to customize the services available in the AmI environment. This goal is achieved by producing new fuzzy rules that correspond to personalized control activities of the involved components. In particular, for each actuator plugged in the AmI network a software agent implements the adaptive algorithm that produces the corresponding FML control code. The overall process is based on two basic functionalities: *learning mode* and *control mode*. In learning mode, the algorithm captures the ambient features representing its inputs in order to generate a fuzzy controller coded in FML. In control mode, the fuzzy controller obtained in the learning mode and associated to a specific actuator, is executed using the AmI Multi-Agents System, as shown in figure 7. These two phases are managed by two independent agent classes, so distributed and concurrent processing is achieved, even though some coordination is required, as better discussed later. First, it is necessary to defined the feature space on which the adaptive layer works on. The features, captured in learning mode, correspond to user's actions (for example, temperature setting) added with the status of AmI environment. In particular, each actuator (for instance, a lighting switch) plugged in the network, communicates its status change in accordance to the user's action and, each sensor (for instance, a lighting sensensor), communicates its actual status in order to model the environment status when the user's action occurred.

Formally, the AmI feature is composed by two parts, as shown in figure 6, (1) AmI environment status matrix; (2) user's actions vector. In particular, the i^{th} column of matrix represents the environment status captured during i^{th} user's action execution which is represented by the i^{th} entry of user's action vector. The size of the user's action vector (the number of columns of AmI environment status matrix) reflects the required completeness of feature space necessary to trigger the learning mode functionality. In practice, the size depends on the specific application.

The adaptive algorithm can be sketched in 7 steps, as shown in listing 2. We refer to the approach described in [Hong96], that we choose as generic model to generate fuzzy rules, but different approaches can be used [Delgado98] [Chen2004].

Essentially, first we generate the feature space relates to the actuator device (step 1), then the user's actions are transformed into clusters (representing similar actions) and fuzzy

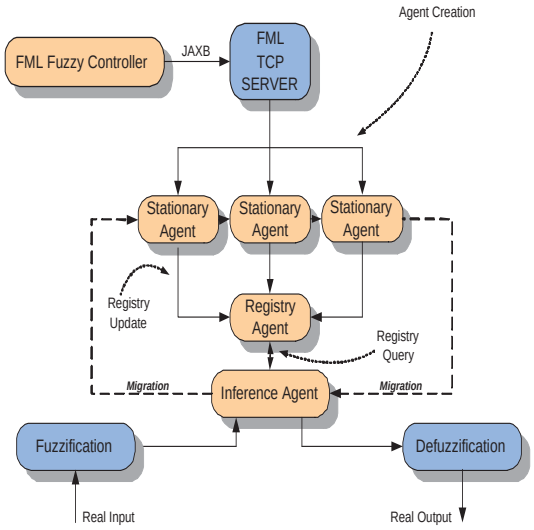

Figure. 7: Fuzzy Multi-Agent System Framework

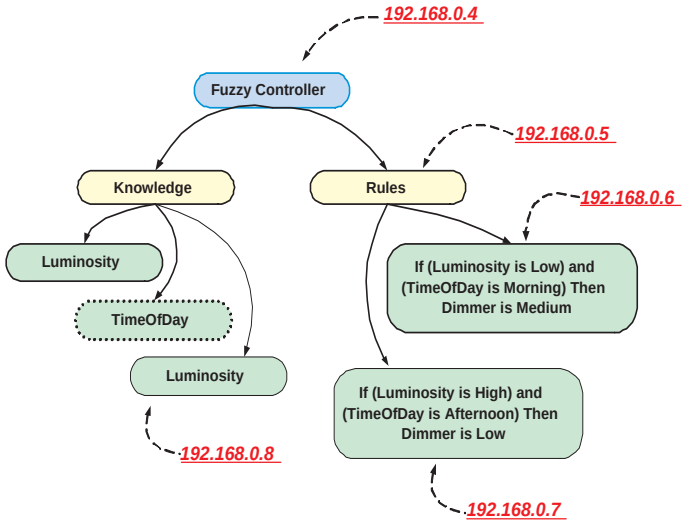

Figure. 4: Distributed Fuzzy Control Tree

$$\left(\begin{array}{cccccccc} temp_1 & temp_2 & \dots & \dots & \dots & temp_{\#data-1} & temp_{\#data} \\ lux_1 & lux_2 & \dots & \dots & \dots & lux_{\#data-1} & lux_{\#data} \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ presence_1 & presence_2 & \dots & \dots & \dots & presence_{\#data-1} & presence_{\#data} \\ extTemp_1 & extTemp_2 & \dots & \dots & \dots & extTemp_{\#data-1} & extTemp_{\#data} \end{array}\right)$$
$$\left(\begin{array}{ccccccc} userAction_1 & userAction_2 & \dots & \dots & \dots & userAction_{\#data-1} & userAction_{\#data} \end{array}\right)$$

Figure. 6: Fuzzy Induction Rules Algorithm Feature

sets are extracted for each cluster (step 2).

```
do {

step 1.
    do {
        update temperature feature space (tfs)
    } while ( # tfs is not meaningful )

step 2.
    Cluster and Fuzzify the
    user actions vector;

step 3.
    Construct initial membership functions
    for AmI status matrix;

step 4.
    rulebase = Construct the initial decision
    table;

step 5.
    rulebase = Simplify the initial decision
    table;

step 6.
    FMLRuleBase = toFML(rulebase);

step 7.
    Start new Agent
        ControlMode(FMLRuleBase);

} while ( true )
```

Listing 2: Adaptive Fuzzy Induction Algorithm

Similar actions are detected by standard derivation (σ_s) of difference values between adjacent data $(diff_i)$ inside the ordered user's action vector. In particular the i^{th} similarity value is:

$$s_i = \begin{cases} 1 - \frac{diff_i}{C * \sigma_s} & \text{for } diff_i \leq C * \sigma_s \\ 0 & \text{otherwise} \end{cases}$$

where C is a control parameter used to modify the shape of membership functions of similarity. From similarity values it is possible to cluster user's actions applying the α - cut method:

If $s_i < \alpha$ then divide the two adjacent data into different group;
else put it in the same group.

Starting from the obtained clusters, it is possible to compute the fuzzy set related to user's actions. The fuzzy shape used to model such action is a triangular fuzzy shape defined by means of (a, b, c) triangle vertex points. In particular, we assume that the center point b lies at COG (center-of-gravity) of the group with membership value at 1. Next, we have to find the membership values of two boundary user's actions in the group, where boundary user's actions mean the minimum and maximum user's actions in the group. The two extreme points a and c of the output membership fuzzy set can be found through the extrapolation of b and the two boundary values. In particular, if $u_i, u_{i+1}, \ldots, u_k$ are the ordered user's actions in j^{th} group, then the central vertex b_j in this group is defined as:

$$b_j = \frac{u_i * s_i + u_{i+1} * \frac{s_i + s_{i+1}}{2} + \ldots + u_{k-1} * \frac{s_{k-2} + s_{k-1}}{2} + u_k * s_{k-1}}{s_i + \frac{s_i + s_{i+1}}{2} + \frac{s_{i+1} + s_{i+2}}{2} + \frac{s_{k-2} + s_{k-1}}{2} + s_{k-1}}$$

The minimum similarity value in the group is chosen as the membership value of the two boundary points u_i and u_k. Then, the following formulas are used to calculate $\mu(u_i)$ and $\mu(u_k)$, where μ_j is the membership functions of the j^{th} group:

$$\mu_j(u_i) = \mu_j(u_k) = min(s_i, s_{i+1}, \ldots, s_{k-1})$$

Now, it is possible to compute the a and c vertex points through interpolation procedure, considering the points $(b_j, 1)$, $(u_i, \mu_j(u_i))$ and $(u_i, \mu_j(u_i))$:

$$a = b_j - \frac{b_j - u_i}{1 - \mu_j(u_i)};$$

$$c = b_j + \frac{u_k - b_j}{1 - \mu_j(u_k)};$$

Starting from the obtained user's action membership functions it is possible to derive the related FML Knowledge Base as shown in listing 3, where $R_1, R_2, \ldots, R_l$ represent the obtained clusters and $a_{R_1}, b_{R_1}, c_{R_1}, a_{R_2}, b_{R_2}, c_{R_2}, \ldots, a_{R_{l-1}}, b_{R_{l-1}}, c_{R_{l-1}}, a_{R_l}, b_{R_l}, c_{R_l}$ are the parameters representing the clusters.
Once computed the fuzzy sets and membership functions able to model the user behavior in AmI environment, it is necessary to fuzzify the environment status matrix in order to obtain the input scenario. First action is to compose the fuzzy set (triangle (a, b, c)), for each row of the input matrix. Since we assume $ba = cb =$ the smallest predefined unit, let a_0 be the smallest value for the element characterizing the row of the input matrix, and a_n be the biggest value. With $a_i - a_{i-1} = a_{i+1} - a_i =$ the smallest predefined unit, we are able to identify the fuzzy set. A typical initial membership function is showed in figure 8.

```
<FUZZYVARIABLE
    domainleft = "a_R1" domainright = "c_Rl"
    ip = "localhost" name = "output"
    scale = "undefined" type = "OUTPUT">
    <FUZZYTERM name="R1">
        <TRIANGULARSHAPE
            param1 = "a_R1"
            param2 = "b_R1">
```

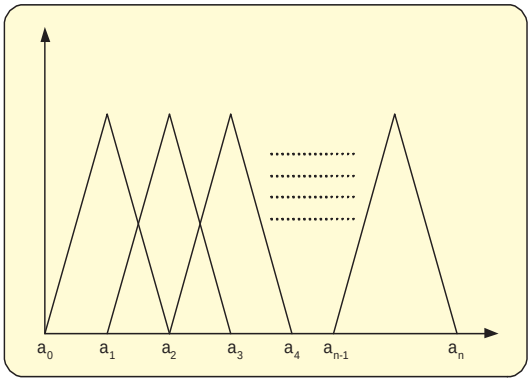

Figure. 8: Typical Membership Function

```
                param3 = "cR1">
            </TRIANGULARSHAPE>
        </FUZZYTERM>
        <FUZZYTERM name="R2">
            <TRIANGULARSHAPE
                param1 = "aR2"
                param2 = "bR2">
                param3 = "cR2">
            </TRIANGULARSHAPE>
        </FUZZYTERM>
        ...
        <FUZZYTERM name="Rl">
            <TRIANGULARSHAPE
                param1 = "aRl"
                param2 = "bRl">
                param3 = "cRl">
            </TRIANGULARSHAPE>
        </FUZZYTERM>
</FUZZYVARIABLE>
...
```

Listing 3: FML Adaptive Knowledge Base

The next step describes the construction of the *decision table*. In particular, we build a multi-dimensional decision table (each dimension represents a corresponding AmI status entity, i.e, the inputs of fuzzy controller under generation) according to the initial membership function. The figure 9 shows an example of a bidimensional initial decision table.
Let $cell$ be defined as the contents of a positions $(d_1, d_2, \ldots, d_m)$ in the decision table, where m is the dimension of table and d_i is the position value. Each cell in table may be empty, or may contain a fuzzy region of the user's action space. For sake of simplicity, let us suppose $R_1, R_2, \ldots, R_l$ be the clusters returned by the first step. The adaptive algorithm eliminates the redundant and unnecessary relations by deleting rows and/or columns of the decision table; in this way it is possible to minimize the number of rules of AmI controller and hence obtain a compact FML program. The optimization procedure is based on five operations:

Operation 1 If cells in two adjacent columns (or rows) in decision table are the same, it is possible to merge these two columns or rows;

Operation 2 If two cells are the same or if either of them are empty in two adjacent columns (rows) and at least one cell in both the columns (rows) is not empty, then it is possible to merge these two columns (rows) into one;

Operation 3 If all cells in a column (row) are empty and if cells in two adjacent columns (rows) are the same, then it is possible to merge these three columns (or rows) into one;

Operation 4 If all cells in a column (row) are empty and if cells in its two adjacent columns (rows) are the same or either of them is empty, then it is possible to merge these three columns (rows) into one;

Operation 5 If all cells in a column (row) are empty and if all the nonempty cells in column (row) to its left have the same region, and all the non-empty cells in the column (row) to its right have the same region, but one different from the previously mentioned region, then it is possible to merge these three columns into two parts.

Applying these operations, the overall number of membership functions is reduced. This deletion may affect the coherence between decision table and input fuzzy memberships. An example of final decision table is showed in figure 10.
Obviously, the algorithm can be adapted to multidimensional decision tables in a simple way.
In order to assure the coherence, it is necessary to assess the membership functions related to the affected dimensions of the decision table. In particular, for operations 1 and 2, if (a_i, b_i, c_i) and (a_j, b_j, c_j) are the membership functions for the i^{th} attribute and d_i and d_{i+1} are the corresponding position values, then the new membership function is $(a_i, (b_i + b_j)/2, c_j)$. For operation 3 and 4, if (a_i, b_i, c_i), (a_j, b_j, c_j) are the membership functions for the i^{th} attribute and d_{i-1}, d_i, d_{i+1} are the corresponding position values, then the new membership function is $(a_i, (b_i+b_j+b_k)/2, c_k)$. For operation 5, if (a_i, b_i, c_i), (a_j, b_j, c_j) and (a_k, b_k, c_k) are the membership functions for the i^{th} attribute and d_{i-1}, d_i and d_{i+1} are the corresponding position values, the new membership functions are (a_i, b_i, c_i) and (a_k, b_k, c_k). At this point, it is possible to derive decision rules from the reduced decision table. Let $cell_{(d_1,d_2,\ldots,d_m)} = R_i$ be a generic entry of decision table. Using such entry it is possible to derive the rule:

If $input_1 = d_1$ **and** $input_2 = d_2$ **and** ... **and** $input_m = d_m$
Then $output = R_i$

This rule, and the whole generated fuzzy rule base, can be modeled using the FML as shown in the listing 4.

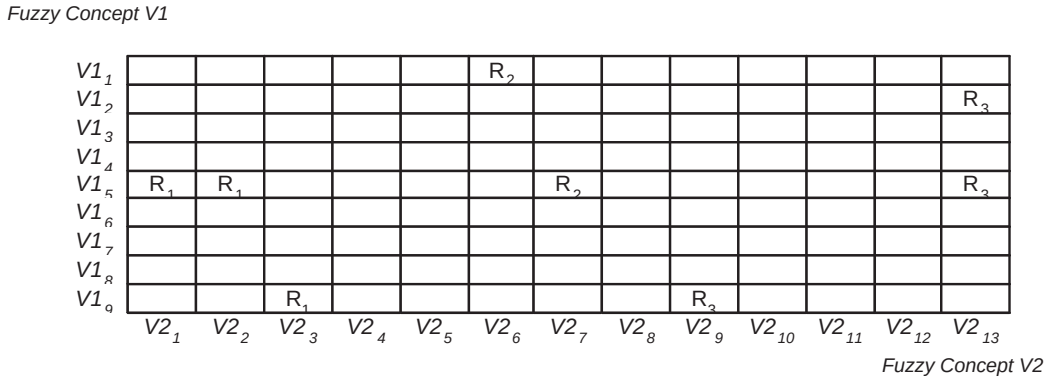

Figure. 9: Initial Decision Table

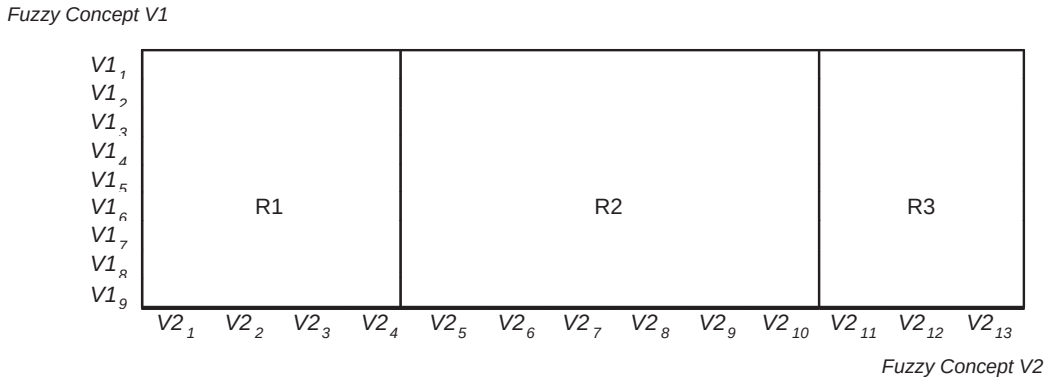

Figure. 10: Final Decision Table

```
<RULEBASE>
    inferenceengine = "MINMAXMINMAMDANI"
    ip = "localhost"

    <RULE
        connector = "AND"
        ip = "localhost"
        weight = "1">
        <ANTECEDENT>
            <CLAUSE not = "FALSE">
                <VARIABLE> input_1 </VARIABLE>
                <TERM> d_1 </TERM>
            </CLAUSE>
            <CLAUSE not = "FALSE">
                <VARIABLE> input_2 </VARIABLE>
                <TERM> d_2 </TERM>
            </CLAUSE>
            ...
            <CLAUSE not = "FALSE">
                <VARIABLE> input_m </VARIABLE>
                <TERM> d_m </TERM>
            </CLAUSE>
        </ANTECEDENT>
        <CONSEQUENT>
            <CLAUSE not = "FALSE">
                <VARIABLE> output </VARIABLE>
                <TERM> R_i </TERM>
            </CLAUSE>
        </CONSEQUENT>
    </RULE>
    ...
</RULEBASE>
```

Listing 4: FML Adaptive Knowledge Base

The $input_i$ parameters, used in listing 4, represent the input fuzzy variable related to i^{th} entry of decision table. These variables have to be defined by FML programmer. The fuzzy variable named $output$ is related to fuzzy concept R_i obtained during the phase 1 of adaptive algorithm.
The obtained FML output is, obviously, a model of centralized fuzzy controller because each ip parameters of each FML tag is initialized with **"localhost"**. In order to create a distributed fuzzy controller achieving the performance of figure 5, the adaptive algorithm uses a list of Internet addresses, used in cyclic way, in order to distribute the rule base in a uniform way obtaining the aforesaid performance.

IV. Conclusions

In this work we reported our experience in mixing soft computing approaches with TCP/IP protocol and agent technology, that represented the basis of a general architecture suitable for AmI systems. Device independence and transparency is achieved thanks to a fuzzy-oriented markup language (FML) able to manage fuzzy concepts, fuzzy rules and fuzzy inference engine directly. Adaptivity strategy is applied in order to dynamically reconfigure the domotic services according to new user's habits and status of home sensors.

References

[1] E. Aarts, Ambient Intelligence: A Multimedia Perspective, IEEE Multimedia, pp. 12-18, 2004.

[2] M. Ditze, G. Kamper, I. Jahnich, R. Bernhardi-Grisson, Service-based access to distributed embedded devices through the open service gateway, Industrial Informatics, 2004. INDIN '04. 2004 2nd IEEE International Conference on 24-26 June 2004 Page(s):493 - 498

[3] K.J. Astrom, Adaptive feedback control Proceedings of the IEEE Volume 75, Issue 2, Feb. 1987 Page(s):185 - 217

[4] B. Schilit, M. Theimer, Disseminating Active Map Information to Mobile Hosts. IEEE Network, 8, No. 5, pp. 22-32, 1994.

[5] G. Acampora, V. Loia, Using Fuzzy Technology in Ambient Intelligence Environment, FUZZIEEE 2005, May 2005, Reno(Nevada), USA, IEEE Press.

[6] G. Acampora, V.Loia, Fuzzy Control Interoperability and Scalability for Adaptive Domotic Framework, IEEE Transactions on Industrial Informatics, vol.1, no.2, May 2005.

[7] XSLT Requirements Version 2.0, W3C Working Draft 14/02/01 http://www.w3.org/TR/2001/WD-xslt20req-20010214.

[8] Extensible Stylesheet Language (XSL) Version 1.1., W3C Working Draft 17/12/03. http://www.w3.org/TR/2003/WD-xsl11-20031217/.

[9] L. A. Zadeh, Fuzzy sets, Information and Control, 8:338-353, 1965.

[10] E. H. Mamdani, Applications of fuzzy algorithms for simple dynamic plants, Proc. of IEE , vol. 121, pp.1585-1588, 1974.

[11] B. DuCharme,XML: Annotated Specification, Prentice Hall, 1999.

[12] T. Takagi, M. Sugeno, Fuzzy identification of systems and its applications to modeling and control, IEEE Trans. Systems, Man and Cybernetics 15(1): 116132,1985.

[13] Yuhui Shi, Combinations of evolutionary algorithms and fuzzy systems: a survey, Fuzzy Information Processing Society, 1999. NAFIPS. 18th International Conference of the North American 10-12 June 1999 Page(s):610 - 614

[14] M.Delgado, A.F.Gomez-Skarmeta, F.Martin, A methodology to model fuzzy systems using fuzzy clustering in a rapid-prototyping approach, Fuzzy Sets and Systems 97: 287-301, 1998.

[15] Min-You Chen, D.A. Linkens, Rule-base self-generation and semplification for data-driven fuzzy models, Fuzzy Sets and Systems 142: 243-265, 2004.

[16] T.P.Hong, C.Y.Lee, Induction of fuzzy rules and membership functions from training examples, Fuzzy Sets and Systems 84: 33-47, 1996.

Authors' Biographies

Giovanni Acampora received the master degree in Computer Science from the University of Salerno in 2003. He is completing his PhD program at University of Salerno working in the design of an Ambient Intelligence environment by using fuzzy control, evolutionary methodologies, and agent paradigm.

Vincenzo Loia received the PhD in Computer Science from the University of Paris VI, France in 1989, and the bachelor degree in Computer Science from the University of Salerno in 1984. From 1989 he is Faculty member at the University of Salerno where he teaches Operating Systems and Adaptive Hybrid Agents . His current position is as Full Professor of Computer Science at Department of Mathematics and Computer Science.
He was principal investigator in a number of industrial R&D projects and in academic research projects. He is author of over 100 original research papers in international journals, book chapters, and in international conference proceedings. He edited three research books around agent technology, Internet, and soft computing methodologies. He is co-founder of the Soft Computing Laboratory and founder of the Multi Agent Systems Laboratory, both at the Department of Mathematics and Computer Science.
His current research interests focus on merging soft computing and agent technology to design technologically complex environments. Dr. Loia is chair of the Task Force " Intelligent Agents" of Emergent Technologies Technical Committee (ETTC), IEEE Computational Intelligence Society.