An Open Integrated Environment for Transparent Fuzzy Agents Design

Giovanni Acampora, Vincenzo Loia
Department of Mathematics and Computer Science, University of Salerno
via Ponte don Melillo, 84084 Fisciano, Salerno, Italy
{gacampora,loia}@unisa.it

Abstract. Recently, computational agents received significant attention in computer science research community. In fact, intelligent agents is a powerful artificial intelligence technology showing considerable promise as a new paradigm for mainstream software development and able to offer new ways of abstraction, decomposition, and organization that fit well with our natural view of the world. However, despite their promise, intelligent agents are still scarce in the market place. A key reason for this is that developing intelligent agent software requires significant training and skill. Artificial Intelligence methodologies and computer networking tools represent the necessary basic knowledge to design and implement advanced agents oriented systems. This papers introduces an integrated development environment supporting the agents developers to design fuzzy-based agents in a simple and fast way. Proposed framework has been realized by integration of theoretical methodologies as fuzzy logic and labeled tree, together with OSS tools as JaxMe2.

1 Introduction

Multi-Agent systems (MASs) consist of a large number of computational agents, interconnected by communication devices, that seamlessly work together in order to achieve prefixed common goals. Different methodologies and techniques have to be exploited in order to design advanced multi-agent systems, ranging from social science, business models, network architectures, up to human interaction design. This methodologies integration allows to design and implement agents-based environment characterized by many features [1]. However, in spite of hard exploitation of computational agents in several domains of computer applications, MAS developers need of significant training, skills and experiences to design and develop advanced MAS-based frameworks. In fact, as previously depicted, MAS can be considered as a composition of different computer science backgrounds, mainly, Artificial Intelligence and Computer Networking. This paper introduces a novel design framework (FuzzyIDE), based on transparent fuzzy agents[2], which allows the system designers to express their ideas in fast and simple way achieving the properties of MAS without additional effort. In particular, this paper introduces a novel tree-based representation for fuzzy controllers allowing the designer to draw their ideas in a natural way and, at the same time, to split the whole controller in several subcontrollers by exploiting the subtree concept. Proposed framework has been realized by integration of theoretical and applicative methodologies as labeled tree and fuzzy control theory.

2 Transparent Fuzzy Control

Fuzzy control theory [3] can be considered as the most widely used application of fuzzy logic [4] [5] From a high level point of view, a Fuzzy Logic Controller (FLC) is an adequate methodology for designing and developing controllers capable of supplying high quality performance in environments characterized by high level of uncertainty and imprecision. However, in spite of these unquestionable advantages, the real design of FLCs is strongly depends upon hardware architecture which will implements the running version of designed controller.

2.1 FLC Labeled Tree

In this section we discuss the labeled tree structure[6] which will be exploited to define the FLC Labeled Tree, the tree-oriented data structure modeling FLCs and representing the core element of *Transparent Fuzzy Control*. A labeled tree is a tree where each node is associated with a label. More formally: Let Σ_V be finite alphabets of vertex labels and edge labels, respectively. Let *V* be a finite nonempty set of vertices, *l* a total function $l: V \rightarrow$, *E* a set of unordered distinct vertices called edges, and *a* a total function $a: E \rightarrow$. Then *G = (V, l, E, a)* is a *labeled graph*. A *labeled tree* is a connected, acyclic labeled graph. Labeled tree labels may be constants, node variables (corresponding to any node value), or path variables (corresponding to any path). Constants corresponds to element, attribute and text values populating XML models. In details, nodes labeled with text values are called text nodes and they are always leaf nodes. Attribute nodes can have only one child node, a text node. Also, any two attributes of a given element cannot have the same label. Element nodes can have zero or more child nodes that can be elements, attributes, or text nodes. Usually, the labels related to element nodes, attribute nodes and text nodes are denoted, respectively, with n_i, a_i and *pcdata*. Then, in the case of XML document modeling: $\Sigma_V = \cup_i \{n_i\} \cup \cup_i \{a_i\} \cup pcd$ and $\Sigma_E =$. In order to represent an FLC through labeled tree structures, it is necessary to individuate the different components of a FLC, to map them on appropriate values for the labels n_i, a_i and *pcdata* and, at the same time, to establish the opportune father-child relationships among different tree nodes. This analysis will result in an acyclic labeled graph, i.e, in a labeled tree. From a bottom-up point of view, a FLC can be view as a collection of fuzzy concepts and fuzzy rules composing, respectively, the fuzzy knowledge base and fuzzy rule base. Classic samples of fuzzy concepts are: temperature, pressure, speed, etc.. Each concept is defined, mainly, by means of: a *name*, a *universe of discourse* and a collection of fuzzy terms. Each fuzzy term can be represented through a pair *(Fuzzy Set, Linguistic Expression)*. By considering the a typical temperature fuzzy concept the following labels can be found:

$n_{FC_1} = FuzzyVariable, n_{FC_2} = n_{FC_3} = n_{FC_4} = FuzzyTe \qquad n_{FC_5} = TriangularSh$

$a_{FC_1} = a_{FC_5} = name, a_{FC_2} = domainL$

$a_{FC_3} = domainRight, a_{FC_4} = scale, a_{FC_6} = not, a_{FC_7} = param1, a_{FC_8} =$ $param2, a_{FC_9} = param3$

...

with:

$$\Sigma_{V_{FC}} = \{n_{FC_1}, n_{FC_2}, n_{FC_3}, n_{FC_4}, n_{FC_5}, a_{FC_1}, a_{FC_2}, a_{FC_3}, a_{FC_3}, a_{FC_4}, a_{FC_5}, a_{FC_7}, a_{FC_8}, a_{FC_9}, \ldots\}$$

and

$$E_{FC} = \{(n_{FC_1}, a_{FC_1}), (n_{FC_1}, a_{FC_2}), (n_{FC_1}, a_{FC_3}), (n_{FC_1}, a_{FC_4}), (n_{FC_1}, n_{FC_2}$$
$$(n_{FC_2}, n_{FC_5}), (n_{FC_2}, a_{FC_1}), (n_{FC_2}, a_{FC_6}), (n_{FC_5}, a_{FC_7}), (n_{FC_5}, a_{FC_8}), (n_{FC_5}, a_{FC_9})$$

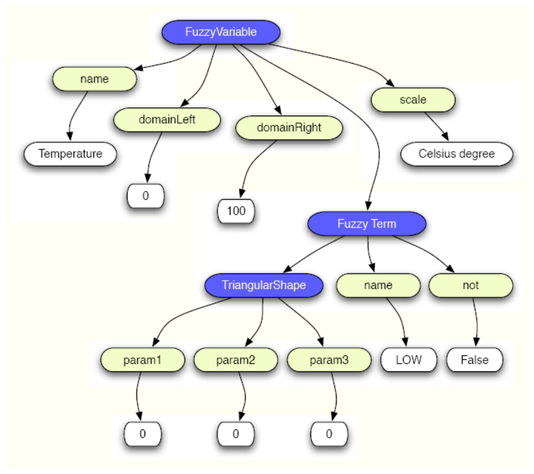


Fig. 1. Fuzzy Variable labeled tree

The *pcdata* information identifies the leaves of XML tree (white tree nodes), i.e., they are strongly depending upon real controller and, for this reason, they are not considered in previous analysis. Figure 1 shows the complete labeled tree derived from previous definitions. Finally, a fuzzy knowledge base can be modeled through a labeled tree whose labels are: $\Sigma_{KB} = \cup_i \Sigma_{V_{FC_i}} \cup$ with the following father-child relationships: $E_{KB} = \cup_i E_{FC_i} \cup \{(n_{KB}, n_{FC_1}), (n_{KB}, n_{FC_2}),$, where I is the label collection modeling i^{th} fuzzy concept and n_{KB} is a dummy root label joining together the different fuzzy concepts sub trees. At same way, focusing on fuzzy rule base, a fuzzy rule is a composition of fuzzy antecedent and consequent parts and, recursively, the antecedents and consequents can be characterized by a sequence of fuzzy clauses, where, each fuzzy clause, is represented through a pair *(FuzzyV ariable, FuzzyTerm)* (in Mamdani case). In short, the labeled tree modeling the fuzzy rule base is: $\Sigma_{RB} = \cup_i E_{FR_i} \cup \{(n_{RB}, n_{FR_1}), (n_{RB}, n_{FR_n}),$ with the following father-child relationships: $E_{RB} = \cup_i E_{RB_i} \cup \{(n_{RB}, n_{RB_1}), (n_{RB}, n_{RB_n}),$ where Σ is the label collection modeling i^{th} fuzzy rule and n_{RB} is a dummy root label joining together the different fuzzy rule sub trees. At the end each fuzzy controller can be modeled by means of a labeled tree populating by following labels: $\Sigma_{FLC} = \Sigma_{RB} \cup \Sigma_{KB} \cup \{n_i$ and $E_{FLC} = E_{KB} \cup E_{RB} \cup \{(n_{FLC}, n_{RB}), (n_{FLC}, n_K,$ where n_{FLC} is a dummy node joining the fuzzy rule base and knowledge base.

3.A Novel Abstract Representation for Transparent Fuzzy Agent Design

The XML representation of FLC allows to model the controller in a human-readable and hardware independent way, i.e., through XML is possible to implement the same FLC on different hardware without additional design and implementation steps. The XML-based language modeling FLCs is named Fuzzy Markup Language(FML) and its fundamentals items, the tags and attributes, are the labels belonging to Σ set.

3.1 Fuzzy Markup Language Definition

The FLC labeled tree is an theoretical representation of a fuzzy logic controller. In order to move FLC tree model to a computer application context, it is necessary to define an opportune computer language grammar mapping the set of labels and relationships defined into the FLC labeled tree. *Document Type Definition* and *XML Schema* are used to define the FML grammar[7][8].

3.2 FLC Labeled Tree and FML in Multi-Agent Systems

The distribution of tasks and resources is both one of the major domains of multi-agent systems. In the FLC design could be very interesting and useful to distribute the fuzzy inference on different computing resources in order to speed up fuzzy computation and to localize rule subset 'near' their data sources. In order to allow this fuzzy distribution, FLC has to be considered as a set of distributed disjoint rule set evaluated in parallel fashion by software agents, hosted on different computing devices, and able to compute appropriate fuzzy operators. In order to fix our transparent FLC view in a multi-agent environment it is necessary to show how FLC tree can be broken in different subtasks, each one representing a portion of initial fuzzy rule base. Let $\Sigma_{RB} = \cup_i E_{FR_i} \cup \{(n_{RB}, n_{FR_1}), (n_{RB}, n_{FR_n}),$ be the rule base of our fuzzy controller and *n* the number of rules. For instance, this rule set can be simply splitted into two different rule base as follows:

$$\Sigma_{RB}^{1} = | \quad | E_{FR_i} \cup \{ (n_{RB}, n_{FR_1}), (n_{RB}, n_{FR_3}), \ldots \}$$

$$\Sigma_{RB}^{2} = | \quad | E_{FR_i} \cup \{ (n_{RB}, n_{FR_{\underline{n}}}), (n_{RB}, n_{FR_{n..}}), \ldots \}$$

This splitting produces two different controllers each one dealing with half of rules contained into . The proposed schema allows to create a fuzzy agency characterized by following agents categories: Splitter agents and Fuzzy agents. The *Splitter agent* is devoted to create a collection of disjoint rule set by following the previous schema; the *Fuzzy agent* is a software entity able to compute the fuzzy inference operator on a rule partition coming from Splitter agent. More in details, if *n* is the cardinality of initial rule base and *m* is the number of fuzzy agents populating the agency then the splitter agent move at most □*n/m*□ rules on each of fuzzy agents.

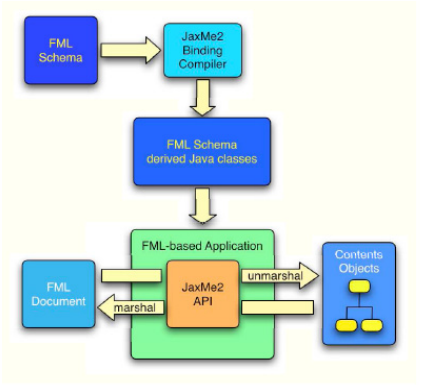


Fig. 3. JaxMe2/FML/Java binding

3.3 Implementing the FML Fuzzy Agents

The FML codes represent only an human-oriented and hardware-independent representation of a fuzzy controller, i.e., the FML programs cannot be computed in a direct way. JaxMe2 represents a direct way to compile and compute the FML services. In fact, the JaxMe2 allows to translate the XML tree structure (in our case, the FOM) into a Java classes hierarchy in direct and simple way through the JaxMe2 compiler. Specifically, JaxMe2 can generate Java classes from XML schemas by means of a JaxMe2 binding compiler. The JaxMe2 binding compiler takes XML schemas as input, and then generates a package of Java classes and interfaces, which reflect the rules defined in the source schema. The JaxMe2 binding framework provides methods for unmarshalling XML instance documents into Java content trees, a hierarchy of Java data objects that represent the source XML data, and for marshalling Java content trees back into XML instance documents. The

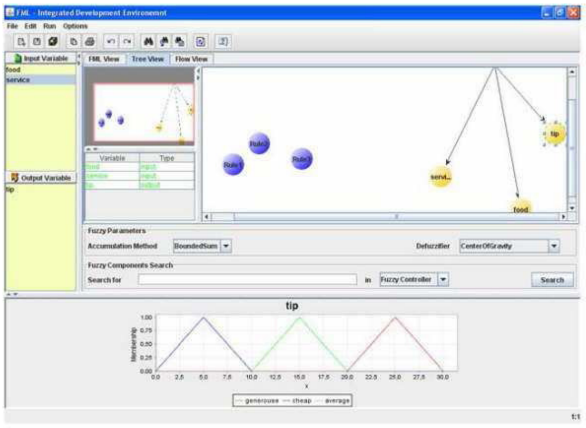
JaxMe2/FML/Java binding is depicted in figure 3. In order to complete the Java representation of FML fuzzy controllers, a fuzzy wrapper class, named FML-Controller has been coded. In particular, FMLController class exhibits a set of methods able to apply the appropriate fuzzy operators to the information derived from JaxMe2 objects. In particular, FMLController constructors allow to create a new fuzzy controller by using the unmarshall method of JaxMe2-API independently from FML file location (file system or network).

4 Integrated Development Environment for Transparent Fuzzy Agent Design

The tree representation of a FLC and, consequently, its mapping in FML language offers an additional important benefit: it allows to design and implement a fuzzy controller by means of simple visual steps. In other words, the FLC tree nature allow the designer to draw a controller through drag and drop actions, i.e., at the same way of visual programming languages allow to write computer programs only making use of the mouse device. By exploiting this benefit an advanced integrated development environment for FLC design and implementation has been realized Its main features are: 1) Fuzzy controller drawing; 2) FML modeling, 3) FML controller synthesis. Fuzzy controller drawing and FML modeling have been discussed previously. FML controller synthesis is a further benefit coming from XML nature of Fuzzy Markup Language. In fact, proposed application exploits the JaxMe2 tools in order to translate the FML abstract controller view in a real implementation based on Java computer languages. Figure 5 shows a screenshot of realized IDE application (FuzzyIDE). Through proposed approach, the MAS developer can define the agent intelligence simply drawing a fuzzy tree and, successively, they can 'compile' this tree obtaining an computable object.

5 Conclusion

In this paper, several principles for structuring systems support for smart multi-agent environments have been presented. Precisely, a novel model for fuzzy controller has been realized; this new representation, based on labeled tree structure, allows to design transparent fuzzy controller, i.e., control systems able to be reprogrammed on different hardware without repeating the design and implementation steps. Moreover, the recursive nature of labeled tree allows to embed the transparent fuzzy controllers in a distributed environment (multi-agent system) in a direct way . The joint use of labeled tree and multi-agent environment has allowed the implementation of an advanced graphical environment allowing the fuzzy designers to draw their ideas by means of distributed fuzzy controllers achieving a drastic decrease of control systems development time.



Variable	Type
food	input
service	input
tip	output

Fuzzy Parameters
Accumulation Method: BoundedSum
Defuzzifier: CenterOfGravity

Fuzzy Components Search
Search for [ ] in Fuzzy Controller

Fig. 4. FML IDE Screenshot

References

1. Ferber J (1999), Multi-Agent Systems: An Introduction to Distributed Artificial Intelligence, Addison Wesley.

2. Acampora G, Loia V (2005) Fuzzy control interoperability and scalability for adaptive domotic framework, *IEEE Transactions on Industrial Informatics*, Vol. 1, No. 2, pp. 97–111.

3. Mamdani EH (1974) Applications of fuzzy algorithms for simple dynamic plants, in *Proc. IEE*, vol. 121, pp. 1585–1588.

4. Zadeh LA (1965) Fuzzy set, *Inform. Control* vol. 8, pp. 338–353, 1965.

5. Hagras H (2007) Type-2 FLCs: A New Generation of Fuzzy Controllers, *IEEE Computational Intelligence Magazine*, Vol. 2, pp. 30–44.

6. Wang YL, Chen HC, Liu WK (1996) A Parallel Algorithm for Constructing a Labeled Tree, *IEEE Transactions on Parallel and Distributed Systems*, Vol. 8, pp. 1236–1240.

7. Acampora G, Loia V (2004) Fuzzy control interoperability for adaptive domotic framework, *IEEE International Conference on Industrial Informatics*, pp. 184–189

8. Acampora G, Loia V (2006) Ubiquitous Fuzzy Computing in Open Ambient Intelligence Environments, *IEEE International Conference on Fuzzy Systems*, Vancouver, Canada, pp. 465–470