A Modular Fuzzy Expert System Architecture for Data and Event Streams Processing

Jean-Philippe Poli, Laurence Boudet

A Modular Fuzzy Expert System Architecture For Data and Event Streams Processing

Jean-Philippe Poli, Laurence Boudet
CEA, LIST, Data Analysis and System Intelligence Laboratory,
91191 Gif-sur-Yvette cedex, France.

Abstract

In many decision making scenarios, fuzzy expert systems have been useful to deduce a more conceptual knowledge from data. With the emergence of the Internet of Things and the growing presence of cloud-based architectures, it is necessary to improve fuzzy expert systems to support higher level operators, large rule bases and an abundant flow of inputs.

In this paper, we present a modular fuzzy expert system which takes data or event streams in input and which outputs decisions on the fly. Its architecture relies on both a graph-based representation of the rule base and the cooperation of four customizable modules. Stress tests regarding the number of rules have been carried out to characterize its efficiency.

Keywords: Fuzzy expert system, complex event processing, data stream processing, rule base representation, policies

1 Introduction

The emergence of connected objects and of the Internet of Things is leading towards a continuous data acquisition from different devices and sensors. Before this recent phenomenon, the data were stored in data warehouse, queried at once and manipulated by algorithms as a whole. With such data in motion, the use cases have changed: for instance, new database management paradigms are introduced, special efforts are made on data compression to avoid networks overload, and supervised or unsupervised learning algorithms are rethought.

Cugola and Margara [8] define the *Information Flow Processing* (IFP) domain as the domain of tools capable of processing information as it flows. Usually, the flow is coming from multiple sources and processed in order to extract relevant knowledge. They also distinguish two subdomains: *Complex Event Processing* (CEP) and *Data Stream Processing* (DSP). On the one hand, DSP consists in processing data flows and in producing a new data flow as output. The Federal Standard defines a data stream as a "sequence of digitally encoded signals used to represent information in transmission". Algorithms for processing such data have to be fast and incremental [21]. In [12], the authors are

revealing the open challenges which must be addressed in the domain of data stream mining, including privacy issues, developing a methodology for stream preprocessing, developing online monitoring systems and balancing resources. On the other hand, CEP differs by the type of data items it considers: an item is a notification of event [15]. CEP aims at managing thousands of events per second [23], for instance, up to 125000 events for a financial software [22]. In this domain, processing mainly consists in filtering, gathering and combining those events in order to build a higher level information [2]. In many real world cases, DSP and CEP have became usual considerations. We can cite for example: system monitoring and fault detection, home automation, security and finance [8].

Whatever the type of items in the stream, i.e. either data or events, the information may be incomplete and imprecise by nature [3]. For instance, sensors may be out of order or inaccurate, and data may be noisy. Fuzzy logic [24] has been specifically designed to mathematically represent uncertainty and vagueness and is a popular tool for dealing with imprecision in many real world problems. Taking advantage of fuzzy logic, fuzzy expert systems allow to easily represent human knowledge about data and phenomena and have been successfully applied to many domains [9, 20].

Comparing with boolean logic expert systems, fuzzy expert systems are often associated with a higher computational cost. Indeed, the whole rule base has to be evaluated in order to compute the outputs, whereas in classical expert systems, a subset of the rules are applied one by one to produce the inference. Moreover, it has been showed that fuzzy rule bases need to be more complicated if only piecewise-linear functions (e.g. trapezoids...) are used instead of non-linear membership functions (e.g. sigmoids...) [5]. Consequently, expensive functions in terms of computation are needed to evaluate the aggregation and the defuzzification [13]. Moreover, in real-world applications, it is possible to have very large rule bases which require a great amount of processor time [1]. Fuzzy controllers have been introduced to overcome these drawbacks and are able to process inputs in real-time [18]. Other papers address the acceleration of fuzzy computation either with dedicated hardware [4] or with the help of Graphics Processing Units (GPU) [10].

To our experience, fuzzy expert softwares which run on CPU platforms are more convenient for many reasons. Firstly, they are easier to interface with an existing system than electronic chipsets. Then, DSP and CEP both rely on software intensive architectures. In terms of scalability, it is possible to use from a single core of a machine to several machines and it can all be done transparently for the user ; for instance, it can take advantage of the virtualization as in cloud-based services. To the best of our knowledge, current fuzzy expert systems, both open source or commercial, rely on straightforward architectures which only manage the classical fuzzy operators (and, or, not). To describe the relations between the data or the events, more sophisticated operators are needed for temporal [7, 16], space [6, 19] or even spatio-temporal [14] reasoning. These operators imply a higher computation cost. Moreover, traditional fuzzy expert systems compute output values only when input values have changed.

This is not compliant with event streams whose events are potentially arriving in an irregular manner: in such a case, expressions may change before the next event arrival (see section 2.3).

Using the terminology of Cugola and Margara, we aim at developing a fuzzy expert system to process information flows, handling the imprecision brought by noisy data, sensors or network problems with fuzzy logic. The motivation of our work is to provide an efficient fuzzy expert system in operational contexts. To enable human experts to author more complex rules, our system is able to efficiently evaluate complex fuzzy relations [16]. To ensure it can interface easily with the various information systems of our partners, we chose to avoid specific architectures (like GPU) and to develop a software for data and event streams processing on regular CPU platforms. Finally, in industrial applications, the efficiency is important not only because there must be a lot of rules, but also because the rules can be applied to a huge number of objects or events per second.

The paper is structured as follows: section 2 describes the architecture of our fuzzy expert system. Section 3 presents the protocol and the results of experiments on synthetic data. Finally, section 4 points out the conclusions.

2 Architecture description

During fuzzy inference, when a group of inputs change at a time t, all the rules containing at least one of those inputs have to be reevaluated. In information streams, inputs may change several times per second, or rules must be applied on thousands of incoming events per second ; the evaluation of the whole rule base may thus need a huge computation time. We introduce an architecture which tends to avoid the system saturation.

2.1 Architecture overview

Figure 1 presents the overview of the proposed architecture. The modularity is ensured by a separation of the tasks and a customization provided by the use of policies. A policy is a set of parameters which customize the behavior of each module. The combination of the behaviors of all the modules enable to address a lot of applications and issues : regular or irregular data rate, delay before inference, etc. The architecture is composed of several modules :

- the **active input queue** gathers the input and group them by timestamp,
- the **scheduler** is able to monitor the system (via the operating system) and to decide which inputs group has to be processed,
- the **evaluator** is in charge of the evaluation of the rules,
- the **output change broadcaster** informs the user about outputs changes.

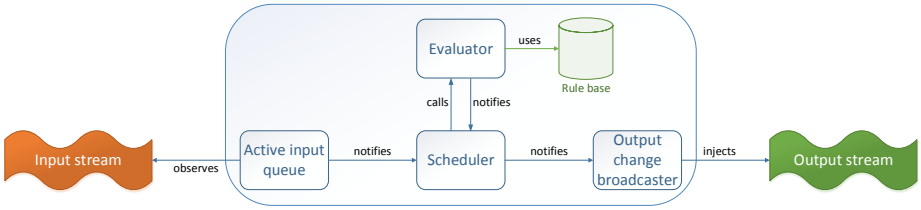


Figure 1: Architecture overview.

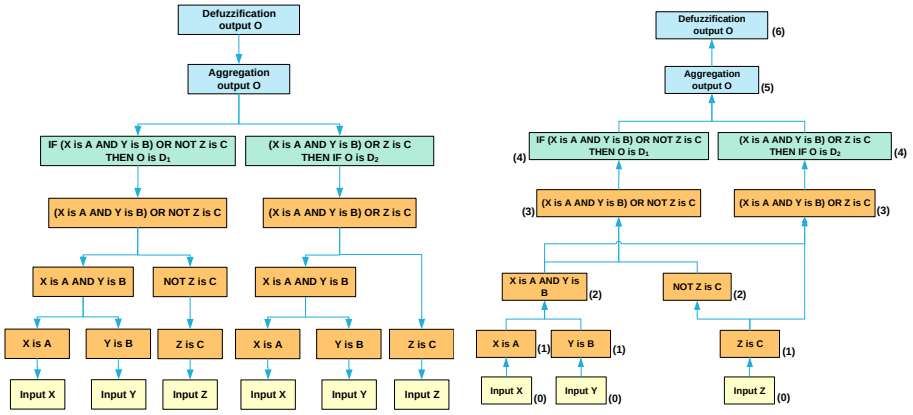
The different modules are supposed to avoid an overload of the system (for instance, the active input queue selects the inputs which should be treated) or user overfeeding (for instance, the output change broadcaster displays only the relevant information). We first introduce how we optimize the rule base representation by common subexpression elimination and the concept of expiration of expressions. We then describe each module of the architecture and give some examples of policies.

2.2 Rule base representation

The rule base in-memory model plays a major role in the efficiency of the fuzzy expert system. Expressions are usually modeled with a tree [17], as in Figure 2(a). However, some expressions can be included in several rules or other expressions: thus, in a tree representation, it is difficult to check the redundancy of such expressions, and it is necessary to evaluate them several times when a group of inputs changed. This problem is known as common subexpression elimination (CSE).

To address the CSE problem in our architecture, we chose to represent each expression by a unique node: thus, the rule base is not represented by a tree anymore but by a graph (figure 2(b)). More precisely, we use an acyclic directed graph to avoid loops during the evaluation. In the graph, an edge $A \longrightarrow B$ means that if the value of the node A changes, it affects the node B and B has to be evaluated again. A node can represent fuzzy expressions (including fuzzy propositions) or rules, and we consider particular nodes for defuzzification and aggregation. Thus, the changes propagate from input nodes to output nodes. The propagation stops if there are no changes during the evaluation of the current node.

The propagation is achieved as particular breadth-first traversal of the graph. However, for a fuzzy n-ary expression, it is necessary to evaluate its n predecessors before its own evaluation, otherwise it would be evaluated n times, and worst, at a certain time, its value would be inconsistent. To avoid this effect, we added a priority information to the nodes. Before starting the fuzzy inference engine, the graph is traversed and a recursive function $priority : Node \rightarrow integer$ is applied. Let N be the current node to be treated, the function $priority$ is defined as follow:



(a) Tree-based representation (b) Graph-based representation

Figure 2: Representations of a base of two rules.

- if N is an input node, then $priority(N) = 0$,
- otherwise, let s_i be the successors of N, $priority(N) = \max_i(priority(s_i))+1$.

Let X, Y and Z be three input linguistic variables, and A, B, C a term from respectively X, Y, Z. Let D_1 and D_2 be two terms of an output linguistic variable O. Then, the rule base is composed of two rules:

- `IF (`X `is` A `AND` Y `is` B`) OR NOT` Z `is` C `THEN` O `is` D_1,
- `IF (`X `is` A `AND` Y `is` B`) OR` Z `is` C `THEN` O `is` D_2.

In Figure 2(b), numbers in brackets represent the evaluation priority of each node; the three inputs are at the bottom of the figure and have a null priority, which means they need to be evaluated first. We will develop in section 2.6 the use of the priority during evaluation.

To the best of our knowledge, current fuzzy expert system does not implement CSE. This is due to the fact that they only use classical fuzzy logic operators which are really fast to evaluate.

2.3 Expiration

Among the sophisticated relations we have implemented, temporal operators [16] and those which depend from them need a special attention when applied on event streams. The particularity of event streams is that the system is noticed of events irregularly. For instance, let consider the fact "the temperature was too hot on Monday from 2 am to 3 am". The system has received two events: at 2am, a temperature high enough to activate the fuzzy proposition "the temperature is too hot", and at 3am, a lower temperature such as "the temperature is too

hot" is false. Now, we consider the temporal operator "occurrence" from [7] which indicates that a phenomenon has occurred on a certain scope in the past: for instance, it can express that "the temperature was too hot during the last 24 hours". Until the next Tuesday 3 am, the degree of truth of this occurrence is strictly greater than 0. After 24 hours, its degree of truth equals 0, whereas the system inputs have not changed since Monday 3 am.

Classical fuzzy expert systems cannot perform this trick since they need that inputs change to compute the outputs. We thus introduce in our system the notion of expiration. Some expressions in the rule base are marked as "expirable" and signal to the scheduler (see section 2.5) that they need to be evaluated again. Expirable components must provide an expiration frequency and a set of criteria to stop the expiration. The expiration frequency is a parameter which depends on the application and the set of criteria depend only on the definition of the operator. To implement expiration, the values of all the expressions are stored in memory to allow partial recalculation.

2.4 Active input queue

Sensor networks are a particular case of information stream. Some sensors measure (data stream) and some others detect (event stream), but they usually work in an asynchronous way. Moreover, some delays can appear in such networks. The active input queue is thus in charge of:

- listening to the information stream,
- fetching the interesting values inside,
- grouping those values by timestamp,
- enqueueing those groups of inputs,
- signaling the scheduler that a new group has been enqueued.

Different policies can be conceived for this component. For instance, in some applications, it is necessary to wait for delayed sensors or delayed network packets before signaling the scheduler. Conversely, it can ignore delays and late arrivals, and thus filter these data. It may also be seen as a firewall which protects the scheduler from irrelevant inputs.

2.5 Scheduler

The scheduler has an important role to play in order to limit the delay between the arrival of the data and the decision making. When a new input set is announced, it decides, regarding its own policy, whether it is important to evaluate it immediately, later or not at all.

In the simplest implementation, the scheduler just gets the first element in the active input queue, asks the evaluator to evaluate this group of inputs and gives the results to the broadcaster. With the use of policies, his behavior can

be more sophisticated. For instance, one particular implementation can monitor the system to determine how busy the CPU cores are and to decide whether a group of inputs can be skipped. Moreover, the scheduler implements the expiration. All the expirable components of the rule base whose evaluation has changed are placed in another queue, waiting to expire.

Another implementation may consist in evaluating on different processor cores of the machine. Each core receives a sub-part of the input set. A simple algorithm based on the graph representation of the rule base is used to separate independent inputs on different sub-parts: this is simply achieved by finding connected components of graph with well-known algorithms of graph theory [11].

2.6 Evaluator

The evaluator is the component which evaluates the different expressions and rules in the rule base. For a set of inputs, it gives a particular set of outputs. It also takes advantage of the rule base representation to perform the computation only when necessary.

In order to evaluate the different nodes of the graph representing the rule base, the evaluator traverses the graph in a certain order. To ensure the right order, we use a priority queue Q. The priority queue Q is implemented such as the nodes with the lowest priority are placed first and such as it contains only one occurrence of each node. The general evaluation algorithm is given below:

```
Q ← changed inputs
while Q ≠ ∅ do
  current ← first(Q)
  Q ← dequeue(Q)
  Evaluate(current)
  if current has changed then
    for all child of current do
      Q ← enqueue(Q, child)
    end for
  end if
end while
return all the values
```

The priority is important in some cases. In figure 2(b), the priority queue ensures the node "(X is A AND Y is B) OR Z is C" is evaluated at the right time. It ensures that if several paths lead to the same node N, all nodes on the paths are assessed before N.

In fuzzy logic, different functions can be used for operators (conjunction, disjunction, negation, implication) evaluation. The policies of the evaluator indicate which version of the operators must be used.

2.7 Output change broadcast

The broadcaster is also an important module because it is in charge of building the output stream. The last step is indeed to inform on the fly the calling system or the user that some outputs have changed. The policies are used to determine when and how the outputs have to be broadcast, for instance:

- the changes can be gathered and sent at regular time intervals,
- only outputs which have changed are broadcast with their new value,
- the changes can be sent with a trace of the activated rules.

It may gather information from the graph and the evaluation of its node to build justifications (to explain why the decision has been made).

3 Experiments

The experiments aim at comparing the performances of the evaluation of different rule bases regarding two different policies of the evaluator module :

- full recalculation mode : all the expressions and nodes are reassessed each time an input changes,
- partial recalculation mode : last values of the nodes are kept in memory and are reassessed only when needed.

The second mode is the one on which our architecture relies. Its modularity, through the use of policies, allows to easily switch between the two modes. In both cases, we only consider the graph-based representation of a rule base, whose evaluation is indeed faster than with a tree-based representation.

3.1 Protocol

These experiments have been carried out on artificial rule bases and data sets whose generation is described hereafter. Let $\{v_i\}_{1 \leq i \leq n}$ be n input linguistic variables, each defined by p terms $T_i^1, \ldots, T_i^p$. Let w be an unique output linguistic variable whose terms are $W_1, ..., W_K$. Those input variables combine into rules by the full conjunctive combination principle :

$$\texttt{IF } v_1 \texttt{ is } T_1^{l_1} \texttt{ and } \ldots \texttt{ and } v_n \texttt{ is } T_n^{l_n} \texttt{ THEN } w \texttt{ is } W_k$$

where $T_i^{l_i}$ refers to a term of v_i with $1 \leq l_i \leq p$ and $k = \sum_{i=1}^{n} l_i - n + 1$. Thus, for a given couple $(n,\ p)$, there are p^n possible combinations of those inputs (i.e. rules) and w has $K = n(p-1)+1$ terms.

For the sake of simplicity, the terms $T_i^{l_i}$ of each variable v_i are defined by triangular membership functions on the domain $[0, p+1]$. By construction, the support of each term $T_i^{l_i}$ is $[l_i - 1; l_i + 1]$ and its kernel is $\{l_i\}$. The same

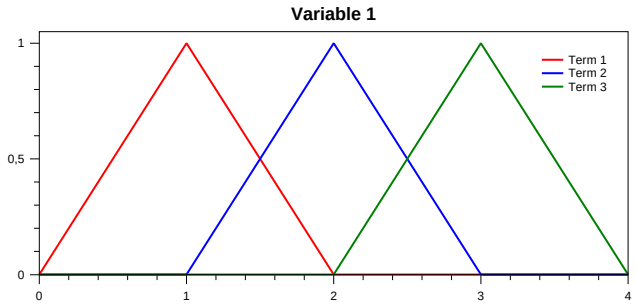


Figure 3: Linguistic variable with 3 terms defined on the domain [0, 4].

construction is used for the terms W_k of w. Figure 3 shows an example of a linguistic variable characterized by 3 terms.

Each input variable v_i receives a data stream of 20 values, which have been generated following an uniform distribution $\mathcal{U}([0, p+1])$.

The architecture has been configured as follows: the active input queue is set in DSP mode, i.e. it waits to receive a value for each input. The scheduler evaluates this group as soon as possible, then the new value of the output is broadcast. This is the most simple configuration of these modules. The two modes of evaluation of the architecture have been obtained by configuring the policy of the evaluator : in one case, it uses its memory functionality; in the other case, it has to compute all the values of the nodes again. The same input data streams have been used for both cases.

Finally, by varying both the number of inputs n and the number of terms p from 2 to 10, we are able to assess the performance of the architecture on large rule bases and to draw some conclusions. Due to the computational cost, the largest configuration was obtained with 6 input variables and 9 linguistic terms. This represents a set of 531441 rules to compute the value of the output w. Even if this is not a realistic case, it is useful to benchmark the proposed system.

3.2 Results

In this section, we first compare the average number of nodes being reevaluated in each mode and then compare the average evaluation time of the rule base. The averages are computed over the 20 values of the data stream in order to decrease the possible biases.

Figure 4 shows the number of evaluated nodes regarding the number of rules, in both modes (full and partial recalculation); the two axes are shown in log-scale. Point clouds confirm the intuition: storing the value of each node allows to stop propagating the changes, and strongly decreases the number of nodes to evaluate. For a rule base with 16807 rules, from $n = 5$ linguistic variables and $p = 7$ terms, 36449 nodes may be evaluated in the full recalculation mode,

whereas in the partial one, only 120 nodes in average are evaluated.

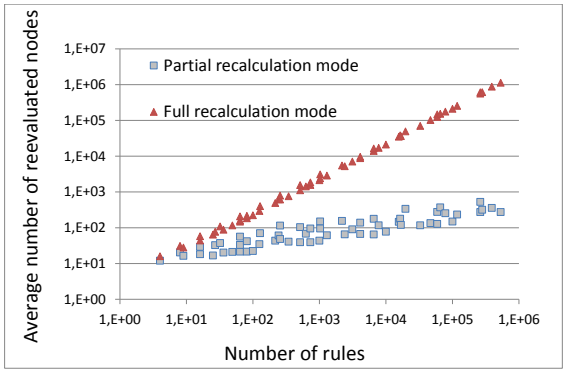


Figure 4: Average number of reevaluated nodes in function of the number of rules (log-scale) for both modes.

The drastic reduction of the number of nodes to be evaluated can be explained by a theoretical analysis. Indeed, the number of nodes N_g of the graph-based representation can be evaluated by the following equation :

$$N_g(n,p) = \underbrace{n}_{inputs} + \underbrace{n \times p}_{propositions} + \underbrace{\sum_{i=2}^{n} p^i}_{conjunctions} + \underbrace{p^n}_{implications} + \underbrace{1}_{aggregation} + \underbrace{1}_{defuzzification}$$

The fuzzy partitions used to create the terms of the linguistic variables explain why, at each time, for each variable, at most 2 terms out of p are activated. Thus, at most $N_g(n,2)$ nodes have to be evaluated: for $n = 5$, at most 109 nodes will be activated. But a large number of them are null because of the conjunctive combination of the inputs. Now, in order to count the number of needed reevaluations, we should consider the worst case : all the active elementary propositions become null, and the same number of propositions get a non-null value. This gives $2 \times N_g(n,2)$ as a pessimistic upper bound of the number of nodes that need to be reevaluated.

Figure 5 shows the duration of the evaluation of the rule bases in both modes. These tests have been processed on only one core of an Intel Xeon X5650 at 2.67GHz on a Windows server. The system is implemented in C#. With the same example as before, to evaluate 16807 rules, full recalculation mode needs approximately 106.8 ms whereas the partial one needs only 17.1 ms, i.e. the latter one is more than 6 times faster than the former one on this rule base structure. It seems that saved computational time is not as high as we could expect considering the saved computations shown just before. But saved computations correspond to the evaluation of a null value by a quite simple function (mainly either by the membership function evaluation or by a conjunctive combination of two expressions) and to affect it to nodes that were already null. Considering this remark, a gain of approximately half an order of

magnitude is a good result by only avoiding the computation of null values of graph nodes.

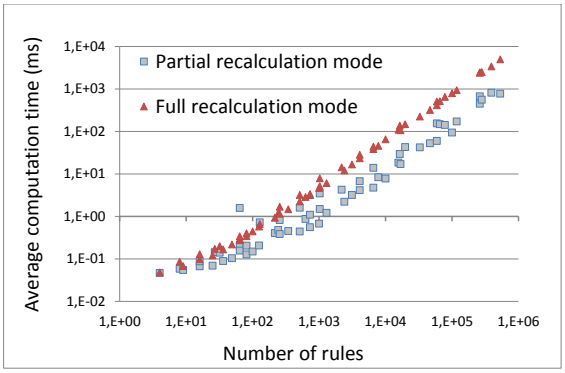


Figure 5: Computation time (in ms) in function of the number of rules (in log scale).

These tests are good stress tests because all the inputs change at the same time. For rule bases of conventional sizes, for instance 300 rules, the engine needs only 0.67ms in the partial recalculation mode. Thus, we can handle inputs which change more than 1000 times per second on only one core.

4 Conclusion

In this paper, we have presented a modern architecture for a fuzzy expert system designed to handle information streams (data streams or event streams). The architecture relies on two aspects. Firstly, the graph representation of the rule base indicates the dependency between inputs, expressions, rules and outputs. Secondly, the use of four cooperating modules permits to filter and to decide when it is possible to process a set of inputs. The introduction of policies in the four modules allows to customize their behaviors regarding the addressed projects or issues.

The described architecture has been implemented and used in several industrial projects in different domains: home automation, decision making in industry and home care services. All projects needed to process either data stream or event stream, sometimes both of them at the same time.

Uncertainty and imprecision are real-world challenges, but others emerge. Users need more fuzzy relations to be able to describe their scenarios or to characterize what they want to extract from the streams. Considering CEP and several thousands of inputs per second, we should parallelize the computations. Finally, online rule base optimization will allow users to sketch first rules and then let the system evolve.

References

[1] Acampora, G., Loia, V.: Fuzzy control interoperability and scalability for adaptive domotic framework. IEEE Transactions on Industrial Informatics 1(2), 97–111 (May 2005)

[2] Alevizos, E., Skarlatidis, A., Artikis, A., Paliouras, G.: Complex event processing under uncertainty: A short survey. In: Fischer, P.M., Alonso, G., Arenas, M., Geerts, F. (eds.) Proceedings of the Workshops of the EDBT/ICDT 2015 Joint Conference. CEUR Workshop Proceedings, vol. 1330, pp. 97–103. CEUR-WS.org (2015)

[3] Artikis, A., Baber, C., Bizarro, P., Canudas-de Wit, C., Etzion, O., Fournier, F., Goulart, P., Howes, A., Lygeros, J., Paliouras, G., Schuster, A., Sharfman, I.: Scalable proactive event-driven decision making. Technology and Society Magazine, IEEE 33(3), 35–41 (Fall 2014)

[4] Basterretxea, K., Del Campo, I.: Scalable Fuzzy Algorithms for Data Management and Analysis: Methods and Design, chap. Electronic Hardware for Fuzzy Computation, pp. 1–30. Information Science Reference (2010)

[5] Basterretxea, K., Tarela, J.M., del Campo I, Bosque, G.: An experimental study on nonlinear function computation for neural/fuzzy hardware design. IEEE Transactions on Neural Networks 18(1), 266–283 (January 2007)

[6] Bloch, I.: Fuzzy spatial relationships for image processing and interpretation: a review. Image and Vision Computing 23(2), 89 – 110 (2005)

[7] Cariñena, P., Bugarín, A., Mucientes, M., Barro, S.: A language for expressing fuzzy temporal rules. Mathware and Soft Computing 7(2-3), 213–227 (2000)

[8] Cugola, G., Margara, A.: Processing flows of information: From data stream to complex event processing. ACM Comput. Surv. 44(3), 15:1–15:62 (Jun 2012)

[9] Garibaldi, J.M.: Do Smart Adaptive Systems Exist? Best Practice for Selection and Combination of Intelligent Methods, chap. Fuzzy expert systems, pp. 105–132. Springer (2005)

[10] Harvey, N., III, R.H.L., Keller, J.M., Anderson, D.: Speedup of fuzzy logic through stream processing on graphics processing units. . IEEE Congress on Evolutionary Computation pp. 3809–3815 (2008)

[11] Hopcroft, J., Tarjan, R.: Algorithm 447: Efficient algorithms for graph manipulation. Commun. ACM 16(6), 372–378 (Jun 1973)

[12] Krempl, G., Žliobaite, I., Brzeziński, D., Hüllermeier, E., Last, M., Lemaire, V., Noack, T., Shaker, A., Sievi, S., Spiliopoulou, M., Stefanowski, J.: Open challenges for data stream mining research. SIGKDD Explor. Newsl. 16(1), 1–10 (Sep 2014)

[13] Laurent, A., Lesot, M.J. (eds.): Scalable Fuzzy Algorithms for Data Management and Analysis: Methods and Design. Information Science Reference (2010)

[14] Le Yaouanc, J.M., Poli, J.P.: A fuzzy spatio-temporal-based approach for activity recognition. In: Advances in Conceptual Modeling, Lecture Notes in Computer Science, vol. 7518, pp. 314–323 (2012)

[15] Luckham, D.C.: The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA (2001)

[16] Poli, J.P., Boudet, L.: Online temporal reasoning for event and data streams processing. 2016 IEEE Conference on Fuzzy Systems, FUZZ-IEEE p. to appear (2016)

[17] Preiss, B.R.: Data Structures and Algorithms with Object-Oriented Design Patterns in Java. Worldwide Series in Computer Science, Wiley (2000)

[18] Reznik, L.: Fuzzy Controllers Handbook. Newnes (1997)

[19] Schockaert, S., Cock, M.D., Kerre, E.: Reasoning about fuzzy temporal and spatial information from the web. World Scientific (2010)

[20] Siler, W., Buckley, J.: Fuzzy expert systems and fuzzy reasoning. Wiley-Interscience (2005)

[21] Silva, J.A., Faria, E.R., Barros, R.C., Hruschka, E.R., Carvalho, A.C.P.L.F.d., Gama, J.a.: Data stream clustering: A survey. ACM Comput. Surv. 46(1), 13:1–13:31 (Jul 2013)

[22] Stonebraker, M., Çetintemel, U., Zdonik, S.: The 8 requirements of real-time stream processing. SIGMOD Rec. 34(4), 42–47 (Dec 2005)

[23] Wu, E., Diao, Y., Rizvi, S.: High-performance complex event processing over streams. In: Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data. pp. 407–418. ACM, New York, NY, USA (2006)

[24] Zadeh, L.: Fuzzy sets. Information and Control 8(3), 338 – 353 (1965)