FUZZY SPECIFICATION IN SOFTWARE ENGINEERING

V. LOPEZ

Faculty of Informatics, Complutense University
Madrid, Spain
** E-mail: ab_vlopez@fdi.ucm.es*
www.fdi.ucm.es

J. MONTERO

Faculty of Mathematics, Complutense University
Madrid, Spain
** E-mail: ab_monty@mat.ucm.es*
www.mat.ucm.es

Judging quality of any decision making procedure is a key problem whenever there is no possibility of developing a sequence of experiments allowing some kind of ratio relative to good results. It may be the case that we have only chances for a unique experiment or no similar experiences are available, but it may be also the case that no standard experiment allows the observation of such a good behavior, simply because such good behavior can not be properly defined. This situation is quite often associated to complex decision making problems. Then the only support we can find for our decision is the decision process itself, the consistency of the arguments leading to such a decision. Checking the quality of such a procedure is therefore a key issue. In this paper we postulate that the design and formal specification of algorithms and processes require a fuzzy approach, since quite often specification is being poorly defined.

Keywords: Fuzzy logic; formal specification; decision making.

1. Introduction

Quite often description of objectives and requirements of a software process are initially poorly defined, but still a procedure can be developed apparently fitting them. In this case, formal specification can not be properly developed with standard techniques. If a set of previous experiences are available or can be developed, additional studies can perhaps allow some statistical procedure in order to evaluate the quality of the procedure. Still, we may have serious problems in case the output needs a linguistic transla-

tion where a probabilistic model may not properly fit (see [1]). In practice, most procedures related to human decision making require ill defined information both in the input and the output. So, we need alternative approaches in order to improve software specification, and more specifically, formal specification of critical algorithms and systems in the software process. As pointed out by Sommerville [2], natural language (NL) is often used to write system requirements specifications as well as user requirements. However, system requirements are more detailed than user requirements, and then NL specifications can be confusing and hard to identify. We can find in the literature formal methods for assuring the right behavior of software, even they are complex sometimes. Formal methods are more popular in critical processes. Secure systems will be an important area for formal methods use [3]. In addition, decision makers use to support their decisions on certain information that is computed by *ad hoc* algorithms (see [5]). Naturally, part of the input data comes from a linguistic framework, perhaps fuzzy relations or fuzzy properties, which should be defuzzyfied before execution. Hence, from the above arguments we can conclude the interest of introducing fuzzy logic methodologies for an accurate formal specification of quite a number of procedures as far as NL of human beings is being involved. In particular, pre and postcondition sentences can be written by means of a propositional logic in which fuzzy sets an fuzzy relations are added.

Documentation methods in software engineering quite often take information in NL, which implies ambiguity and imprecision. Software engineering should in this case try to improve the quality of software by means of diagrams, pictures and a large amount of text. In addition, formal methods, as *Z notation* [9], solve ambiguity by using crisp set theory and mathematical logic. Our proposal is to take it as the initial model and then bring fuzzy logic into specifications in order to develop formalizations for algorithms and software specifications (one of the main benefits of formal specification is its ability to uncover ambiguities that NL produces, so those specifications involving fuzzy constrains and properties should be approached within a fuzzy framework). This paper points out the future relevance of this kind of specifications in software specification of requirements, to become a key element in any software project involving non-technical users.

2. Formal specification of algorithms

An algorithm **A** can be formally described by its formal specification. Such a formal specification describes the computational situation before and after executing an algorithm, by means of relationships between the input and the

output data. Informal specification is expressed in NL, but computations require a previous formalization, quite often involving fuzzy relations, either in the input data or the output data. The following definition generalizes the concept of formal specification [4] of an algorithm in a fuzzy context.

Definition.- Let **A** be an algorithm or a process. The 'formal fuzzy specification' of **A** is given by a triple $(ED, fPre, fPost)$ where

(1) ED is an explanatory database [7], that is, a collection of relations in terms of which the meaning of $fPre$ and $fPost$ are defined;
(2) $fPre$ (fuzzy precondition) is the canonical form of propositions and fuzzy first order formulas in NL that expresses the initial constraints and knowledge about data; and
(3) $fPost$ (fuzzy postcondition) is the canonical form after execution of the algorithm (fuzzy constraint propagation and inference is due, and the postcondition is the formalized goal of the algorithm or the process).

Example 2.1. Let X be a subset of n members in the universe U of people. We pretend to decide if *'most of'* people in X is *'tall'* (for simplicity we shall not develop its explanatory data base). It is easy to understand that modelling this problem in a crisp environment will produce some information lose. A fuzzy interpretation of *'most of'* or *'tall'* is closer to the NL interpretation. The proposal specification in the usual notation becomes:

fun $Fuzzy$ (X: **array**[1..n] **of real** ; μ_{mostof}: [0,1]^n $\rightarrow$ [0,1],
μ_{Tall}: [0,1] $\rightarrow$ [0,1]) **ret** b: **[0,1]**;
{fPre: $n \geq 0$}
{fPost: $b = \mu_{mostof}(\mu_{Tall}(x_1), \ldots, \mu_{Tall}(x_n))$}

Fuzzy specification of algorithms and processes can be therefore understood as a way to write those constrains and requirements that limit a problem. Software specification of requirements involves fuzzy relations as well. Fuzzy sets and fuzzy logic are also powerful tools to increase the quality of the information from software specification. Some technical concepts are explained now in order to understand our proposal to introduce fuzzy logic into formal specification of systems.

3. Fuzzy specification in formal methods

Formal specification of algorithms and processes can be improved by introducing fuzzy logic into formal methods. Formal methods are useful to produce documentation free of ambiguity and imprecision in crisp specification of systems. Our main objective is to generalize formal methods into

a fuzzy framework. First of all we introduce some concepts in which formal methods are founded.

A formal language **L** is a language defined by precise mathematical formulas. Any language **L** can be viewed as a subset of **A***, where **A*** denotes the set of all words over alphabet **A**. A formal language must be formally specified, for example as strings produced by some formal grammar, strings described by a regular expression, or strings accepted by a finite automata between others. Given a formal language in terms of usual sets, operations between them will produce other formal languages. Concatenation, intersection, union, complement and Kleene star are some of the most useful operations for making new languages. Automata theory in general is a good reference for this topic, see [8]. A specification language is a formal language used during system analysis, requirements analysis and design in a software project. The main goal of a specification language is to describe the system at a much higher level than a programming language, nearer to natural language from where comes all information about the system. Specification language is used for describing what but not how, so implementation details are not suitable. There are two types of specification languages: property-oriented and model-oriented. In the property-oriented approach, specifications of programs consist of logical axioms in a logical system with equality, describing the properties that the functions are required to satisfy. CAST or Common Algebraic Specification Language is a good sample based on first-order logic. On the other hand, model-oriented specifications consist of a realization of the required behavior. *Z notation* is one of this formal specification languages. It is based on axiomatic set theory, first-order predicate logic and lambda calculus. In any case, specifications must be refinement before be implemented. Formal specification is a mathematical description of software (or hardware). It may be used to develop implementation of software and it describes what the system should do. Developed design and code will depend on a previous good formal specification of the system. After that, formal verification is used for proving the correctness of algorithms with respect to the formal specification, using formal methods of mathematics, proving theorems concerning properties that specification shows. Formal methods are based upon mathematics and they can be used to produce documentation in which information is written in a good level of abstraction. Those documents are free of ambiguity and imprecision. However, information is translated in a crisp way before producing documentation. Our goal here is to take information from natural language (in fuzzy terms) and apply formal methods to specify formally

the system. *Z notation* [9,10] is a particular formal method based upon set theory and mathematical logic. This makes *Z notation* the ideal formal method to introduce fuzzy logic in formal specifications.

4. Z notation and fuzzy logic

The goal of *Z notation*, like other formal methods, is to *'add precision to aid understanding'* [9]. *Z notation* is defined in [9] as a mathematical language with a powerful structuring mechanism, that in combination with natural language can be used to produce formal specifications. *Z notation* offers mechanisms that look very adaptable to a fuzzy framework. A collection of linguistic labels can be defined as a 'Free Type'. Free types in *Z notation* are used to model enumerated collections, so we can define

$$Class_3 ::= label << 0..2 >>$$

and give names to the four elements of the set *Class*:

$$\begin{array}{|c} \text{bad, regular, good} : Class_3 \\ \hline \text{bad} = \text{label } 0 \\ \text{regular} = \text{label } 1 \\ \text{good} = \text{label } 2 \end{array}$$

Here 'bad', 'regular' and 'good' are linguistic labels that we also can assign to a μ function $(\mu_{bad}, \mu_{regular}, \mu_{good})$ defined as relations an also we can then define relations between them, like ordering:

$$\begin{array}{|c} \leq_{label} : Class_3 \leftrightarrow Class_3 \\ \hline \forall l_1, l_2 : Class_3 \bullet l_1 \leq_{label} l_2 \Leftrightarrow \varphi \end{array}$$

where φ is a first order logic formula or a fuzzy constraint.

Mathematical objects and properties will be collected together in '*schemas*'. A '*scheme*' in *Z notation* is a pattern of declaration and constraint. It consists of two parts: a declaration of variables, and a predicate constraining their values. A scheme is denoted in the following way

$$Name \widehat{=} [declaration | predicate]$$

Every specification in the fuzzy framework can be formalized in *Z notation*. In particular, it is possible to define the types used in the system, by means of *schemes*, as we show in the following example.

Example 4.1. A real estate uses a software system to keep data about estates and make valuations about them. It is easy to understand that coexistence of fuzzy and crisp relations is mandatory because of several factors as brightness, views, size, etc. The object '*Estate*' is defined in general by means of its identification name and a list of characteristics.

$$ESTATE = (ID, [Neighbourhood, Light, Views, Rooms, Size, ...])$$

where fuzzy characteristics as *'Neighbourhood'*, *'Light'* and *'Views'* are fuzzy granules in the sense of [7], and there are also crisp characteristics like number of rooms. Taking into account that fuzzy characteristics are going to be computed by means of fuzzy sets, we can formalize the following types in *Z notation*:

- *Estate* is a record that contains some characteristics, we show some:

$$Estate \widehat{=} [Id, rooms : \mathbb{N}; size : \mathbb{R}; light, views : Granule | size > 0]$$

- *Granule* define a fuzzy granule in the sense of [7]. Its formal specification in Z notation is:

$$Granule \widehat{=} [k : \mathbb{N}; label : LingLable[0..k] |$$

$$(k mod 2 = 0) \wedge (\forall i \in \{0..k\} \exists \mu_i : Trapezoidal \bullet label_i \leftrightarrow \mu_i) \wedge$$

$$(\forall i \in \{0..k-1\} label_i \prec label_{i+1})]$$

- *Trapezoidal* defines a fuzzy set by means of a μ function. It can be formalized as:

$$Trapezoidal \widehat{=} [\mu : \mathbb{R} \mapsto [0,1]; a, b, c, d : \mathbb{R} |$$

$$(0 \leq a < b \leq c < d) \wedge \mu(x) = \begin{cases} 0 & (x < a) \vee (x > d) \\ \frac{x-a}{b-a} & x \in [a, b] \\ 1 & x \in [b, c] \\ \frac{d-x}{d-c} & x \in [c, d] \end{cases}]$$

- The subtype *Triangular* of *Trapezoidal*, is a useful type:

$$Triangular \subseteq Trapezoidal | c = d$$

Z notation provides also schemas for declarations, predicates or renaming. The information contained in schemas may be combined by conjunction, disjunction, negation, quantification and composition. This application of the schema language allows to represents a state of the system as an object of the corresponding schema type, and make reasoning about computing states during a process or an algorithm.

5. Conclusions and future work

The main conclusion of this paper should be the need for developing a new fuzzy specification of algorithms and processes adding computer with words techniques into formal methods, in particular in early stages of the process, just when the specification is 'customer-oriented' and NL is used. In particular, in this paper we have shown how specification of algorithms, pre and post-conditions can be improved if they allow fuzzy relations and constraints. In this way, software specification can take information from NL, that includes a lot of fuzzy concepts and perceptions. Granulation and CW take into account the existence of fuzzyness within specifications, so we can avoid lose of information due to lack of accuracy of alternative crisp approaches. In addition, the model gets closer to non-technical users.

As an immediate future work, we pretend to develop formal specification of fuzzy types that will permit us to make reasoning and relate computing states in algorithms an states of the system in process looking for improved designs, processes, algorithms or projets in which fuzzy logic must be considered.

References

1. J. Montero and M. Mendel, *Crisp acts, fuzzy decisions*, (In S. Barro *et al.*, eds., Advances in Fuzzy Logic, Santiago de Compostela, 1998, pp. 219–238).
2. I. Sommerville, *Software engineering*, (Addison-Wesley Publishers, 2004).
3. A. Hall, R. Chapman, *Correctness by construction: developing a commercially secure system*, (IEEE Software, 2002, 19(1): 18–25).
4. C.A.R. Hoare, "An axiomatic basis for computer programming," *Communications ACM*, 12:89–100, 1969.
5. J. Montero, V. López and D. Gómez, *The role of fuzziness in decision making*, (In D. Ruan *et al.*, eds.,Fuzzy Logic: an spectrum of applied and theoretical issues, Springer, 2006, in press).
6. J. Woodcock, J. Davis, *Using Z. Specification, refinement and proof*, (Prentice Hall, 1997).
7. L.A. Zadeh, *From computing with numbers to computing with words- From manipulation of measurements to manipulation of perceptions*, (Ed. Paul P. Wang, Computing with words, Willey, 35–68, 2001).
8. Hopcroft, J. E., Ullman, J. D., *Introduction to Automata Theory and Computation*, (Addison-Wesley, 1979).
9. Woodcock, J. and Davis, J., *Using Z: Specification, refinement and proof*, (Prentice-Hall, 1997).
10. Spivey, J. M., *The Z notation: A reference manual*, (Prentice-Hall, 1998).