Object Oriented Software for Fuzzy Arithmetic

Frantisek Hunka
University of Ostrava, 30. dubna 22, 701 03 Ostrava1, Czech Republic
+420 597 192 175 – frantisek.hunka@osu.cz – www.osu.cz

Abstract
Anticipatory systems can use for its activity not only simulation models but also models based on fuzzy sets mechanism of expressing uncertainty. Fuzzy sets models can be fruitful mainly in solving problems, in which some of the principal sources of uncertainty are nonstatistical in nature. Fuzzy sets are frequently replaced by fuzzy numbers mainly in the engineering applications for much simpler handling. A fuzzy number can be expressed by a number of different forms that are dependent on complex computation that the given forms can bring. There are also a number of described arithmetic approaches for basic fuzzy arithmetic operations. The paper tries to show how object oriented software can be helpful in the implementation of the different methods and different approaches of fuzzy arithmetic.
Keywords : anticipatory systems, fuzzy arithmetic, object oriented modeling, Java, BETA

1 Introduction

Fuzzy arithmetic covers basic arithmetic operations applied on fuzzy numbers. In general, fuzzy number can be expressed by several ways. The possible forms can be triangular form, trapezoidal form or more precise form expressed by explicit function e.g. parabolic function and the last mostly in theory used way is exploiting core property of fuzzy numbers, membership function, expressed either by functional table of values or by explicitly given function.

In the engineering practice, however, the triangular and trapezoidal forms prevail for their relative simplicity. Direct implementation of extended arithmetic operators on fuzzy numbers is computationally complex and is similar to solving a nonlinear programming problem. For this reason either triangular fuzzy numbers (TFN) or trapezoidal fuzzy numbers (TrFN) are used. Unfortunately the TFN shape is not closed under multiplication and division because the result of these operators is polynomial membership function and triangular shape only approximates the actual result.

To solve the problem a parametric representation of fuzzy numbers and their arithmetic operators was proposed and described by the papers ([2],[3]). Together with the proposal the same authors also introduced a new approximation, which is consistent, computationally fast and accurate estimation for arithmetic operations.

The parameterized fuzzy number (PFN) contains further three parameters and its arithmetic operations can be regarded as further extended operators of the same applied on TFNs. The result of fuzzy multiplication and fuzzy division applied on the TFNs is nonlinear operations with a polynomial membership function.

The simplest forms of fuzzy numbers expression we mean triangular and trapezoidal form of fuzzy numbers can be used as anticipation of the more precise arithmetic operations based on parameterized representation of fuzzy numbers. This could be beneficial in better estimation of the further parameters of the more complex expression of fuzzy numbers forms.

The aim of the paper is to show facilities that provide object oriented modeling to cover this application domain. In our approach we focus mainly on two object oriented programming languages Java and BETA and present their possibilities. Viewing the whole problem we may consider PFNs as an incremental extension of TFN. This view covers of course both attribute extension (specification) and arithmetic operation extension. As was showed in other publications e.g. ([6], [8]) object oriented approach can be used with benefit for fuzzy applications.

2 Fuzzy Numbers and Arithmetic

Basic fuzzy number arithmetic operations are defined in the Tab. 1. The operators of fuzzy addition and subtraction are exact defined. However fuzzy multiplication and division operators defined are only defined by approximations to the actual results. In operations addition and subtraction standard approximation can be replaced by actual result, which is found by rewriting the membership function to define a set of closed intervals.

Table 1: Arithmetic operations on TFNs and their definition

Arithmetic Operations	Definition
$\tilde{A} \oplus \tilde{B} =$	$\langle a_1 + a_2, b_1 + b_2, c_1 + c_2 \rangle$
$\tilde{A} \ominus \tilde{B} =$	$\langle a_1 - c_2, b_1 - b_2, c_1 - a_2 \rangle$
$\tilde{A} \otimes \tilde{B} =$	$\langle a_1 \cdot a_2, b_1 \cdot b_2, c_1 \cdot c_2 \rangle$
$\tilde{A} \oslash \tilde{B} =$	$\langle \frac{a_1}{c_2}, \frac{b_1}{b_2}, \frac{c_1}{a_2} \rangle$

Commonly used approximations for the standard operators for TFNs showed that incorrect results could be obtained. The discrete approaches were not a suitable alternative since they did not have a concise representation, required internal storage of many discrete points to reconstruct membership functions, and suffer from computational complexity.

The new approximation designed by ([2], [3]) is built upon six parameters which describe a parameterized fuzzy number. The presentation of a PFN is

$$\tilde{A} \rightarrow \langle a, b, c, \lambda, \rho, n \rangle \tag{1}$$

where a, b and c has the same meaning as with TFNs. λ and ρ parameters are the spread ratios and are defined

$$\lambda_i = \frac{b_i}{a_i}, \ \rho_i = \frac{b_i}{c_i} \tag{2}$$

Spread ratios are included because they characterize the approximation error. The term n is the order of the polynomial expression for the membership function. The definition for using these six parameters for performing arithmetic operations by [2] is shown in the Tab. 2.

Table 2: Fuzzy arithmetic with parameterized fuzzy numbers

Arithmetic Operations	Definition
$\tilde{A} \oplus \tilde{B} =$	$\langle a_1 + a_2, b_1 + b_2, c_1 + c_2, \sqrt[max(n_1,n_2)]{\lambda_1^{n_1} \cdot \lambda_2^{n_2}}, \sqrt[max(n_1,n_2)]{\rho_1^{n_1} \cdot \rho_1^{n_2}}, max(n_1, n_2) \rangle$
$\tilde{A} \ominus \tilde{B} =$	$\langle a_1 - c_2, b_1 - b_2, c_1 - a_2, \sqrt[max(n_1,n_2)]{\lambda_1^{n_1} \frac{1}{\rho_2^{n_2}}}, \sqrt[max(n_1,n_2)]{\rho_1^{n_1} \frac{1}{\lambda_2^{n_2}}}, max(n_1, n_2) \rangle$
$\tilde{A} \otimes \tilde{B} =$	$\langle a_1 \cdot a_2, b_1 \cdot b_2, c_1 \cdot c_2, \sqrt[n_1+n_2]{\lambda_2^{n_1} \lambda_2^{n_2}}, \sqrt[n_1+n_2]{\rho_1^{n_1} \cdot \rho_2^{n_2}}, n_1 + n_2 \rangle$
$\tilde{A} \oplus \tilde{B} =$	$\langle \frac{a_1}{c_2}, \frac{b_1}{b_2}, \frac{c_1}{a_2}, \sqrt[n_1+n_2+1]{\lambda_1^{n_1} (\frac{1}{\rho_2})^{n_2+1}}, \sqrt[n_1+n_2+1]{\rho_1^{n_1} (\frac{1}{\lambda_2})^{n_2+1}}, n_1 + n_2 + 1 \rangle$

It is recognized that the main source of error between the actual and approximated results of fuzzy multiplication and division is the difference between the polynomial shape and the straight line approximation. The new approximation by the paper ([2],[3]) is based on the fact that a better approximation than a straight line approximation is a polynomial approximation. The generalized polynomial is then scaled and added to the original linear result.

The scaling factor times the generalized polynomial is added to standard approximation to obtain a new approximation for each α-cut. For multiplication the α-cut expression are

$$P_{N(L)} = P_L + G(\alpha, n)\tau_L(n\bar{\lambda})(b - a), \tag{3a}$$

$$P_{N(R)} = P_R + G(\alpha, n)\tau_R(n, \bar{\rho})(c - b), \tag{3b}$$

where symbol $\bar{\lambda}$ is a geometric mean of λ. And for the division

$$Q_{N(L)} = D_L + G(\alpha, n)\tau_L(n\bar{\lambda})(b-a), \tag{4a}$$

$$Q_{N(R)} = D_R + G(\alpha, n)\tau_R(n, \bar{\rho})(c-b) \tag{4b}$$

The scaling expressions are for the left segment,

$$\tau(n, \bar{\lambda}) = 0.568\bar{\lambda} + 0.11n - 0859 \tag{5a}$$

and the right segment

$$\tau(n, \bar{\rho}) = -1.85\bar{\rho} + 0.144n + 1.19 \tag{5b}$$

The generalized polynomial expression, which closely tracks the shape of the polynomial for the actual multiplication and division has the following expression

$$G = \left[(-\alpha)^{\frac{(-1)^n-1}{2}} \sum_{i=2}^{n} (-\alpha)^i\right] - \left[\frac{(-1)^n+1}{2}\right]\alpha \quad \text{for } 0 \leqslant \alpha \leqslant 1 \tag{6}$$

In the derived expression there is an important aspect useful for object oriented approach that is that the new approximation introduced by the paper ([2],[3]) is composed from the standard approximation, expressed by e.g., P_L, P_R or D_L, D_R, which is added to a new part. The new part as can be seen from the papers ([1], [3]) is created by multiplication of scaling expression and generalized polynomial expression. The generalized polynomial expression is a function of two arguments and n.

All these aspects may be used in object oriented design of fuzzy arithmetic for fuzzy numbers.

3 General Requirements for Fuzzy Arithmetic Modeling

As mentioned in the previous parts there is a number of different ways of expressing fuzzy numbers and in this way a number of basic arithmetic operation declarations too. One can see that this area is just suitable for object oriented modeling with the stress for incremental development of various fuzzy arithmetic operation declarations. This will call for good modeling facilities of object oriented software. Fuzzy arithmetic can be expressed by classification we mean creation of proper hierarchy of classes (*class/subclass system*) that fulfills fuzzy arithmetic demands. Observing arithmetic operations for triangular FN and Parameterized FN one can see that the part of the operations remains the same and the other part is extended in the Tab. 1 and Tab. 2. This feature can be used for object oriented description in the sense that a method describing a given operation in super class can be further

extended (specialized) in its subclass. In our case triangular FN is the super class and parameterized FN is the subclass.

The same situation can be seen with the operations for new approximation that considerably improves standard approximation of alpha cuts. In addition the new approximation is created by addition of the standard approximation and a given polynomial extension. All is further explained in the book [7] and the paper [4]. Standard approximation might be described in the super class and its extension in the subclass. There are tools in object oriented perspective that enable to create these requirements. Described approach has other advantages, which is implementation of the next classes that express other forms of fuzzy numbers such as trapezoidal FN class and so on.

3.1 Anticipation at the Level of Programming Environment

Some programming environments such as Mjolner BETA System may be considered to be anticipatory systems in the sense that they are able to offer the user only restricted language constructs choice during the development of the program. It may be simply called syntax-directed edition. The choice of the language constructs is derived from the programming language grammar and such programming developing environments with this ability for a given language are generally called *grammar based languages*. Contrary to most of the developing environments provide a graphical user interface, a browser for navigation among classes and their methods, but the program editor is purely textual. The first step towards creating grammar based developing environment is structure based environments, which is characterized by focusing on support for editing. Structure based environments enforce a structured look on programs by only allowing the user to manipulate a program as a tree. This is a step up in abstraction level from the common single character view. The next step in this approach is the recognition of *abstract syntax tree*, rather than parse trees or plain text, as the primary representation of the programs. Abstract syntax trees become a common way to store programs in those environments, and textual representation is merely generated when needed.

In practice it means that editor reminds user immediately when missing some syntactic part of the language. User can not continue without correcting the error. Moreover the editor is hyper structured, which means that only the required part of the edited text may be seen and the other parts may be hidden. The result of this is the more powerful and robust created software application but such an environment can be also used for learning a new language.

3.2 Means of Object Oriented Modeling

Generally basic means for object oriented modeling includes means for composition and classification. Those means enable to determine classification hierarchies and composition hierarchies. In principal there are three different forms of object

composition. The tight form of composition, known as whole part of composition, the loose form of composition, known as aggregation and localization. Whole part composition used to be implemented by static references that are unable to refer on other object than the one declared for the variable. If the language does not proved static references like Java the part objects are instantiated within the constructor of the whole object.

Aggregation is the least problematic issue for implementation as this type of composition is implemented as *dynamic references* and all object oriented languages provides this possibility. Localization, the third form of composition, is a mean for describing (organizing) that the existence of *phenomena/concepts* are restricted to the context of a given phenomenon. That is the local component *phenomena/concepts* are dependant upon the composite phenomenon. Localization as the third form of composition can be applied in the object oriented languages that support block structure and nested class declaration. Both BETA and Java fulfill this requirement.

Classes support classification of objects with the same properties, and subclassing supports the specialization of the general properties. In a subclass it is possible to specialize the general properties defined in the superclass. This can be done by adding data attributes and/or methods. However it is also possible to modify the methods defined in the superclass. Modification can take place in different ways. A method (virtual) may be redefined (overrided) in a subclass. The other way, represented e.g. the BETA language, is that method (virtual) cannot be redefined in a subclass, but it may be further defined by extended definition. Actually this extended method is *submethod* (in the same way as for subclass) of the method defined in the superclass. This implies that the actions of this way of modification are automatically combined with the actions of the extended method in a subclass. This way is of course a bit difficult, since the programmer cannot ignore the action of the superclass method. On the other hand this can be very useful most of applications as the process of development can continue incrementally and can simply absorb all new ideas or intentions.

The Java language provides pseudovariable *super*, which refers to the immediate superclass to the given class. This solution can replace overriding by further extension in a similar way like in BETA but only in simple cases. So its use is restricted and does not provide all Betas comfort and possibilities for modeling. Class of the language that enables incremental modification instead of redefinition is structural compatible with its superclass. Class of the language that uses redefinition of its methods in subclasses is name compatible with its superclass. In the former approach the further definitions do not violate the invariant of the superclass.

Powerful abstraction mechanism of the BETA languages is based on the pattern concept that introduces (in the most general way) the same syntactic declaration both for class and method (of course includes process, exception and so on). This concept enables to extend virtual mechanism from originally use to the classes too and introduces so called virtual classes.

Virtual classes are useful for defining "parameterized" general classes with a strongly typed language. Examples of such classes are containers classes. The decisions about the element of such classes should be deferred to the subclasses of the general class. In the same way as (virtual) method a virtual class makes it possible to defer part of the definition to a subclass. Virtual classes may be seen as an alternative to *generic* types as found in other languages (Java, Ada and Eiffel). On the other hand unique syntactic declaration brings possibility for methods hierarchy (within adequate classes).

As the BETA language has the same syntactic mechanism both for class and method declaration (pattern) they can behave in the similar way. Classes may be virtual and methods may be organized in hierarchies.

Virtual classes may be qualified either locally or globally. The former mentioned possibility could be helpful for fuzzy arithmetic modeling. It means that data attributes can be declared as virtual classes and further extended in subclasses. Possibility of local declaration is very helpful in fuzzy arithmetic modeling, as it is closely connected with the "virtual" method. Locally declared virtual classes enable to follow the actual type of the enclosing class. In this way this enables that part of the virtual method declaration can be used as an invariant to the subclasses.

However Java object oriented language is rather focused on reusability than modeling. As it does not declare so called static variables (do not miss it with the static variable declaration) it can not have means for explicit distinguish between composition and aggregation and other mechanisms based on this feature. All variables in Java except for variables for primitive types are dynamic variables. The other restricted modeling facilities concern "incremental" program development. Java prefers rather overriding than extension of the methods. There is of course keyword super for a reference to a super class method. But the aim for introducing it has a different base from the keyword "inner" used in BETA.

For processing different types of collections Java provides a collection framework, which strictly distinguishes between interface and implementation. It is perfect comfortable for a standard application tasks but less flexible for modeling demands despite the facts that latest version of Java provides generics a mechanism similar to virtual class mechanism provided by the BETA language. In the practical example that accompanies the paper we try to use Java in the way the BETA can be used.

Further BETAs feature dedicated to modeling is so called singular object declaration. In this narrow definition it means that single object in the created system can be created as a singular object. The other notion for this construction can be a classless object. The user does not have to use e.g. singleton pattern for checking that only one instance of a given class is created. In the broader sense this construct can be used for locally defined objects. Together with the concept of localization locally defined classes may be declared in a similar way.

4 Design and Implementation of Fuzzy Arithmetic

Object oriented design of fuzzy arithmetic should cover arithmetic operation, which might be implemented in a different way depending on a concrete class of fuzzy number. For this reason a declaration of an interface including arithmetic operations will be the first step. The interface actually acts as an abstract super class for all other declared classes. FuzzyNumber interface describe arithmetic operations and type of necessary arguments as well as type of return objects from the operations.

Created model should cover both TFNs, PFNs potentially other fuzzy number expression such as trapezoidal fuzzy number but of course it could cover fuzzy numbers expressed by its discrete membership function. As in our paper we are primarily interested in fuzzy by arithmetic all these operation are placed in the common interface called FuzzyNumber. This mechanism enables to implement different operations in a different ways. On the other hand it also facilitates to use further specialization principle for declared operations. It means that operations declared in the super class can be further extended in its subclass.

In object oriented perspective of PFN can be considered as an extension of the TFN in that way that PFN has next specific attributes and the basic arithmetic operation are in this sense also extended. For this reason ParameterizedFN class is the subclass of TriangularFN class. TriangularFN class is also the superclass for all derived classes including fuzzy number expressed by an explicit membership function. The whole structure of designed classes and interface can be seen in the Fig. 1 in the form of UML class diagram.

TriangilarFN class represents object oriented description of triangular fuzzy number. In the class three attributes a, b, c, representing triangular fuzzy number are declared. Apart from declared attributes the class contains implementation of the arithmetic methods described in the Tab. 1.

ParameterizedFN class further extends TriangularFN superclass by declaring next data attributes, λ , ρ and n. Their meaning is explained shortly in the second part of the article or in a more detailed way in the paper [2]. As attributes and express so called spread ratios and can be derived from the attributes a, b and c they are not explicitly entered by the user but they are calculated at object creation (instantiation). The term n is the attribute representing the order of the polynomial expression for the membership function. The arithmetic operations for the parameterizedFN are declared in the Tab. 2. Their implementation is following. The first part of the calculation is the same as for the triangularFN so it is used from the superclass. The other part differs so it is declared as extension in the superclass method.

The other arithmetic operations declared in this class could also include scalar addition and scalar multiplication. Scalar addition represents addition of PFN and a scalar (crisp number) and in the same way scalar multiplication means multiplication of PFN and a scalar (crisp number).

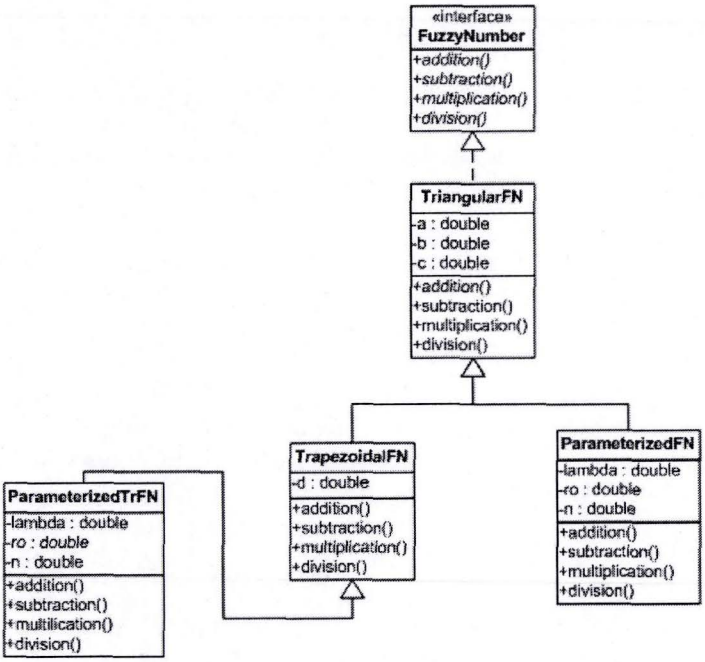

Fig. 1: Fuzzy number arithmetic – UML class diagram

Some applications require using of α-cuts for various calculations. Just using α-cuts declared in ParameterizedFN enable to get far precise results than the use of standard approximation for α-cuts. Calculating error is mainly seen when the multiplication or division arithmetic operation are repeatedly executed. In that case the use of parameterized α-cuts is considerable. It has its reason in the fact that multiplication and division have both polynomial membership function and the α-cuts far better follow actual shape of the membership function.

One of the basic aspects of object oriented approach is information hiding, which means that not all part of the class (attributes and methods) can be "visible" from outside of the class. This is a powerful means that helps check the access inside the class. The possibility of hiding information is used for auxiliary operations such as calculation of the generalized polynomial or polynomial part of the new approximation that are assessable only within the class, not from the outside. On the other hand basic arithmetic methods and left and right segment of the new approximation are accessible both from inside and outside of the given class.

TrapezoidalFN class represents other possible classes derived from the description of the triangular fuzzy number class. We made this decision on the basis of the next additional parameter but we have not done extensive experiments with the class yet as we focused mainly on TriangularFN and ParameterizedFN classes.

5 Results

Implementation of the FuzzyNumber interface and TriangularFN and ParameterizedFN classes was made in Java object oriented language. To prove the implementation is correct we use the similar examples as were used in the papers ([2], [3]). Achieved results for basic arithmetic operations are presented in the Tab. 3. New approximation results in α-cuts form is presented in the Tab. 4.

Table 3: Results from the basic fuzzy arithmetic operations

	a	b	c	λ	ρ	n
operand 1	25.0	40.0	55.0	1.60	0.73	1
operand 2	4.0	10.0	16.00	2.50	0.63	1
addition	29.0	50.0	71.0	4.0	0.45	1
subtraction	21.0	30.0	39.0	2.56	0.29	1
multiplication	100.0	400.0	880.0	2.0	0.67	2
division	1.56	4.0	13.7	1.6	0.002	3

Given two PFNs:

$$\tilde{x} \rightarrow \langle 70, 100, 130, 1 \rangle \tag{3}$$

$$\tilde{y} \rightarrow \langle 4, 10, 16, 1 \rangle \tag{4}$$

Table 4: Comparing actual quotient with new approximation

	Actual quotient		New approximation	
α	Left segm.	Right segm.	Left segm.	Right segm.
1	10.0	10.0	10.0	10.0
0.9	9.2	11.0	9.3	10.9
0.8	8.4	12.0	8.6	12.0
0.7	7.7	13.3	7.9	13.5
0.6	7.1	14.7	7.3	15.3
0.5	6.5	16.4	6.7	17.4
0.4	6.0	18.4	6.2	19.8
0.3	5.6	20.9	5.7	22.5
0.2	5.1	23.8	5.2	25.5
0.1	4.7	27.6	4.8	28.9
0	4.4	32.5	4.4	32.5

6 Conclusion

There are two points the paper focuses on. The first is anticipatory models and their exploiting in object oriented modeling and the other is comparing means for object oriented modeling for fuzzy number arithmetic. Models based on fuzzy numbers may be also used for anticipation nonstatistical sources of uncertainty. As the paper shows anticipatory models may be hidden inside the software development tools as demonstrated with the BETA "syntax directed editor". Such an editor anticipates all possibilities of created code at the given point of the program. It gives the user not only perfected structured view on the code but does not allow to miss semantic end of the blocks and so on. On the other hand the paper also shows the advantage of object oriented approach in modeling complex problems, where there is a number of possibilities (methods) to chose from, depending on the desired purpose. The best way to model this problem is to use incremental development approach, which is in its impact more natural and flexible. In addition it is much closer to our human real world apprehension. Generally one can usually determine that descriptive actions have something in common, which can create actions describe in super classes and further extended in the subclasses. Languages based on reusability can refer its superclass by using pseudovariable super. It is effective but not as effective like an extended approach in general.

There are generally two ways how to implement virtual method mechanism in object oriented languages. One is based on overriding of the superclass method declaration and the other is based on the further extension and specialization. While the former approach seems to be a simpler for the software designer it can leads rather to awkward software constructions. On the other hand these constructions are better to understand on the first sight. The later mentioned approach is a bit demanding on the first sight as the software designer has to keep in mind all methods declarations done in the methods superclasses. On the other hand it is more flexible in the way of further extension and more compact. Of course it is conditioned by the possibilities of having such implementation facilities. Our example is not very large and vast to show the overall problem but even from this example it is visible. The later approach is much closer to out natural comprehension of real word approach as we are not pull out from our thinking

Acknowledgement

The paper was supported by the research scheme of the Institute for Research and Application of Fuzzy Modeling No: MSM6198898701.

References

[1] Dubois Didier, Prade Henry (1978) Operations on fuzzy numbers. Internat. J. Systems Science 9, pp. 613–626.

[2] Giachetti Ronald E., Young Robert E.(1997) A parametric representation of fuzzy numbers and their arithmetic operators. Fuzzy sets and systems. vol 91, No. 2. pp. 185–202.

[3] Giachetti Ronald E., Young Robert E. (1997) Analysis of the error int the standard approximation used for multiplication of triangular and trapezoidal fuzzy numbers and the development of a new approximation. Fuzzy Sets and Systems 91, pp. 1–13.

[4] Hunka Frantisek (2003) Object Oriented Approach in Cluster Analysis. Acta Electronica et Informatica. No. 2, Vol. 3, pp. 55–59.

[5] Kalin Martin (2001) Object-Oriented Programming in Java. Prentice Hall.

[6] Lalonde Wilf, Pugh John (1994) Smalltalk V: Practice and Experience. Prentice Hall. Englewood Cliffs New Jersey.

[7] Madsen Ole L., Moller-Pedersen Birger, Nygaard Kristen (1993) BETA Object-Oriented Programming Language. Addison Wesley.

[8] Schmucker Kurt J. (1984) Fuzzy Sets, Natural Language Computations, and Risk Analysis. Computer Science Press. Rockwille. Maryland