12th International Conference on Industrial and Engineering Applications of AI and Expert Systems, IEA/AIE-99, Cairo, Egypt, 1999

Modelling Fuzzy Sets Using Object-Oriented Techniques

Gary Yat Chung Wong, Hon Wai Chun

City University of Hong Kong
Department of Electronic Engineering
Tat Chee Avenue
Kowloon, Hong Kong
e-mail: eehwchun@cityu.edu.hk

Abstract. This paper describes a new approach to model fuzzy sets using object-oriented programming techniques. Currently, the most frequently used method to model fuzzy sets is by using a pair of arrays to represent the set of ordered pairs, elements and their grades of membership. For continuous fuzzy sets, it is impossible to use infinite number of elements to model, therefore a limited number of array elements are used. Because the grade of membership can be calculated by a membership function, we introduced an approach that models fuzzy set using membership functions directly. Our new approach reduces memory required to model fuzzy sets. Furthermore, grades of membership are calculated dynamically only when needed. Compare with the approach mentioned before, our approach seems to offer advantages in memory space and computation time when modelling systems with complex continuous fuzzy sets.

1 Introduction

In traditional approach, software model to fuzzy set [8, 9] use a pair of array to store the element and grade of membership to represent order pairs [8, 9]:

$$A = \{(x, \mu_A(x)) | x \in X\} , \qquad \textbf{(1)}$$

where $\mu_A(x)$ is the membership function [8, 9] representing the grade of membership of x in fuzzy set A. In other words, it maps the fuzzy set domain X to membership space 0 to 1 for a normalised fuzzy set [9]. It is straightforward to model discrete fuzzy set by use of this method. For continuous fuzzy set, limited number sample of membership function pre-processed and stored as discrete fuzzy set [2, 5]. By using sampled ordered pairs, grades of membership between samples can be found by interpolation. In our approach, we would like to store membership function directly instead of storing discrete ordered pairs by means of object-oriented programming techniques.

In the following Sections, the traditional approach will be described briefly. Then, our new object-oriented approach and its implementation will be shown.

2 Traditional Approach

For discrete fuzzy set, using a pair of array to store ordered pairs [2, 5] is the simplest model. Also, operations on discrete fuzzy set using this model are straightforward. For example, the ordered pairs of fuzzy set A and B are *{(1,0.2), (2,0.5), (3,0.8), (4,1), (5,0.7), (6,0.3)}* and *{(3,0.2), (4,0.4), (5,0.6), (6,0.8), (7,1), (8,1)}* respectively, then $A \cap B=\{(3,0.2), (4,0.4), (5,0.6), (6,0.3)\}$. This intersection operation on discrete fuzzy set in software can be done by the following procedure.

```
1.  Sort fuzzy set $A$ by element
2.  Sort fuzzy set $B$ by element
3.  Expand fuzzy set $A$ by adding element with zero grade of membership,
    which exist in fuzzy set $B$ but not $A$.
4.  Expand fuzzy set $B$ by adding element with zero grade of membership,
    which exist in fuzzy set $A$ but not $B$.
5.  Start intersection operation
    For $k = 0$ to $size(A) - 1$
        $GradeOfMembership = min(\mu_A(x_k), \mu_B(x_k))$
        If $GradeOfMembership \neq 0$
                Add $(x_k, GradeOfMembership)$ to resultant fuzzy set
```

Other fuzzy operations [2, 6, 8, 9] on discrete fuzzy sets can be done by replacing the $min(\mu_A(x_k), \mu_B(x_k))$ function with other functions, such as maximum for union.

For continuous fuzzy set, sampling of membership function is required to initialize fuzzy set. The procedure is shown below:

```
1.  $SamplingPeriod = (DomainMax-DomainMin)/(MaxNumberOf Sample-1)$
2.  $Sample = DomainMin$
3.  While $Sample<DomainMax$
        Add $(Sample, \mu(Sample))$
        $Sample = Sample + SamplingPeriod$
```

According to the domain and number of sampling points, a similar process can be used find the resultant fuzzy set after fuzzy operation. Instead of adding the ordered pair, ordered pair of sample and *operation* $(\mu_A(Sample), \mu_B(Sample))$ is used, where *operation* is a function which process fuzzy set operation. For example, if the operation is intersection, $min\ (\mu_A(Sample), \mu_B(Sample))$ is used instead of $\mu(Sample)$. As shown in the procedure below:

```
1.  $SamplingPeriod = (DomainMax-DomainMin)/(MaxNumberOf Sample-1)$
2.  $Sample = DomainMin$
3.  While $Sample<DomainMax$
        Add $(Sample, min\ (\mu_A(Sample), \mu_B(Sample)))$
        $Sample = Sample + SamplingPeriod$
```

To implement continuous fuzzy set operations using the traditional approach, an "adequate" number of samples should be taken. First, if the sampling period is too large, the membership function itself may have distortion. Second, if the resultant fuzzy set sampling period is not small enough, distortion may occur, even the operand fuzzy sets do not have any distortion. As see the example below.

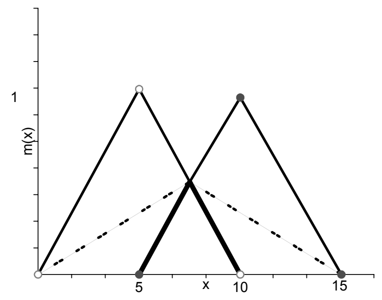

Fig. 2.1. Intersection of two fuzzy sets with triangular membership function which number of sampling is three and domain are *0* to *10* and *5* to *15* respectively. The resultant fuzzy set domain is *0* to *15* and number of sample is three, which is not large enough.

As see from Fig. 2.1, the bold line is the expected result and the dotted line is the result, which do not have enough samples. This problem can be solved by increase the number of sample, however, it is a difficult task to find an "adequate" number of sample because it depends on the membership function. If the number of sampling is too large, it wastes memory. If it is too small, the membership function distorts. We therefore propose an alternative modelling method using object-oriented techniques.

3 Object-Oriented Approach

In this paper, we would like to introduce another approach to model fuzzy sets. According to the first paper in fuzzy set [8] by Professor Zadeh:

A fuzzy set (class) A in X is characterized by a membership (characteristic) function $f_A(x)$ which associate with each point in X a real number in the interval [0, 1], with the value of $f_A(x)$ at x representing the "grade of membership" of x in A.

It is quite natural to describe the above concept using object-oriented programming concepts. We consider fuzzy set as a class that has an attribute to describe the domain space and associated with a "membership function class". This *functor* class contains the actual membership function method, *func*, which maps each point in the domain to the interval *[0, 1]*. As shown in the UML [3, 4] class model in Fig. 3.1.

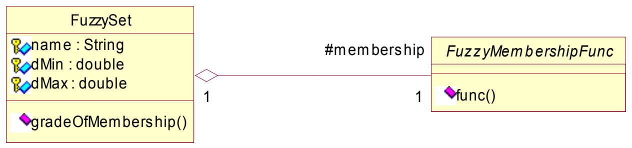

Fig. 3.1. Class diagram of fuzzy set and fuzzy membership function

FuzzyMembershipFunc is an abstract class; all fuzzy membership function classes should inherit from this class and override the *func* abstract method with its own membership function. The class diagram below describes this structure.

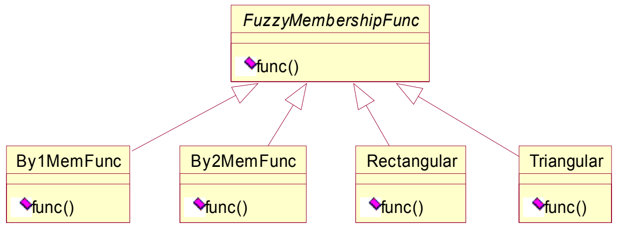

Fig. 3.2. Class diagram of fuzzy membership functions (*By1MemFunc* and *By2MemFunc* will be explained later)

By use of this implementation model, grades of membership are calculated when required. No sampling is needed to initialize fuzzy set because the *func* method defined in the fuzzy membership function classes already captures this relationship. To create a new fuzzy set, just need to assign a suitable fuzzy membership function object to that fuzzy set object.

The UML sequence diagram below shows the process of an application program requesting the grade of membership of a fuzzy set object.

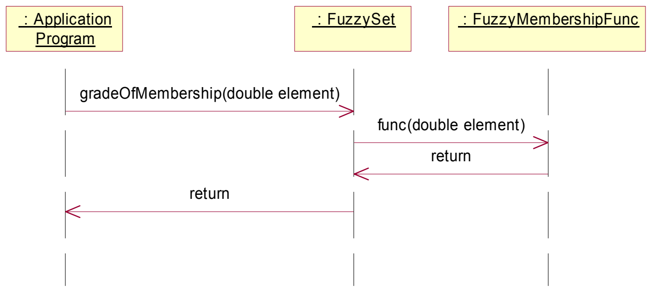

Fig. 3.3. Application program request grade of membership of an element in the fuzzy set

When an application program request the grade of membership of an element, the element is passed to the fuzzy membership function object via the fuzzy set object by calling *gradeOfMembership(element)*, then *func* method in *FuzzyMembershipFunc* invoked and compute the grade of membership in runtime and finally to the application program.

For fuzzy set operations, beside the classes stated above, operator objects, such as complement, alpha cut, maximum, minimum … etc are required. For each operator class, a method is used to describe fuzzy set operation. For example, the method in complement operator class is $f_{A'}(x) = 1 - f_A(x)$ where A' is the resultant fuzzy set. Fig. 3.4 shows the class hierarchy of fuzzy operators.

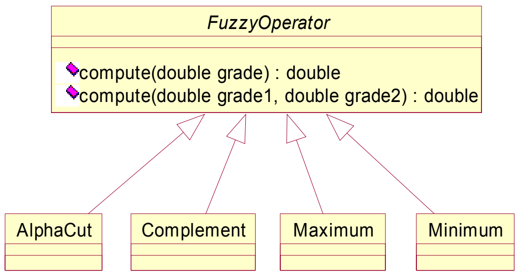
FuzzyOperator
compute(double grade) : double
compute(double grade1, double grade2) : double
AlphaCut
Complement
Maximum
Minimum

Fig. 3.4. Fuzzy operators class hierarchy

There are two *compute* methods in *FuzzyOperator* class, for unary operations and binary operations. *FuzzyOperator* is an abstract class; all operator classes should inherit from this class. Operators for unary operation such as *AlphaCut*, *Complement* … etc need to override the *compute(double grade)* method. Operators for binary operation such as *Maximum*, *Minimum* … etc need to override the *compute(double grade1, double grade2)* method. To create new fuzzy set object after fuzzy operation, beside the operand fuzzy set(s), the membership function associate with it and fuzzy operator object, we also need two special fuzzy membership functions, *By1MemFunc* and *By2MemFunc* for unary and binary operation respectively. In the following paragraphs, we will describe how these two special function work with fuzzy operation. The structure of *By1MemFunc* and *By2MemFunc* are shown in the UML class diagram below:

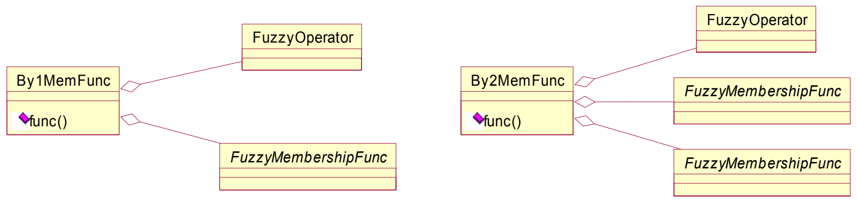

Fig. 3.5. Structure of *By1MemFunc* and *By2MemFunc*

The *FuzzyMembershipFunc* object associate with *By1MemFunc* and *By2MemFunc* object is copied from the membership function object of the operand fuzzy set(s). Fig. 3.6 shows the sequence diagram of unary operation.

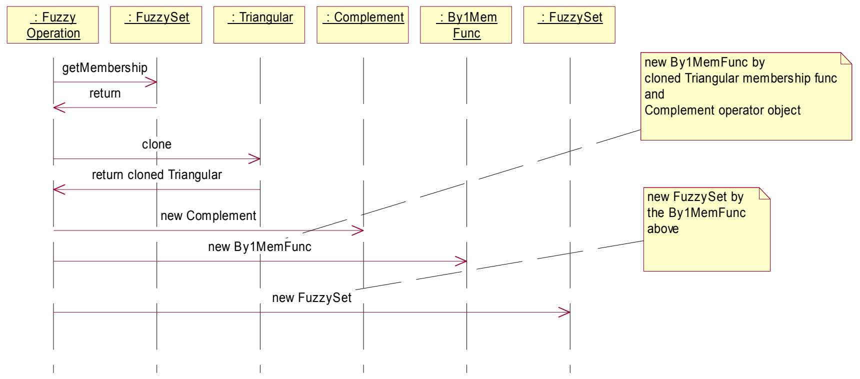

Fig. 3.6. Example: Complement of fuzzy set with triangular membership function

The fuzzy set generated after fuzzy operation in the figure above is different from the fuzzy set in Fig. 3.3. The membership function which associate with this fuzzy set is a *By1MemFunc* which link a cloned triangular object and a fuzzy operator, but the membership function described in Fig. 3.3 do not link any other object and able to return the grade of membership directly. The sequence diagram below shows how the grade of membership of the resultant fuzzy set returned. (All the objects below are generate by the process describe in Fig. 3.6)

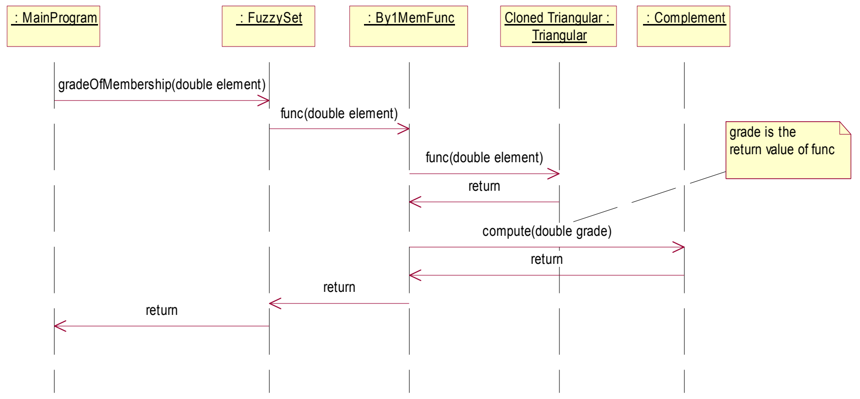

Fig. 3.7. Grade of membership return from the fuzzy set generated by unary operation

When the grade of membership of an element is required from this resultant fuzzy set, the element is passed to the unary fuzzy membership function (*By1MemFunc*) object from fuzzy set object. Then, *By1MemFunc* object obtains the grade of membership of the element from the cloned operand fuzzy membership function object. After that, the grade of membership passed to the operator object. After calculation, the resultant grade of membership obtained and returned to the fuzzy set and to the main program that request grade of membership in runtime.

Binary operation is similar to unary operation (see Fig. 3.6), but it needs to clone one more membership function and use *By2MemFunc* instead of *By1MemFunc*. Fig. 3.8 below shows an example of binary operation.

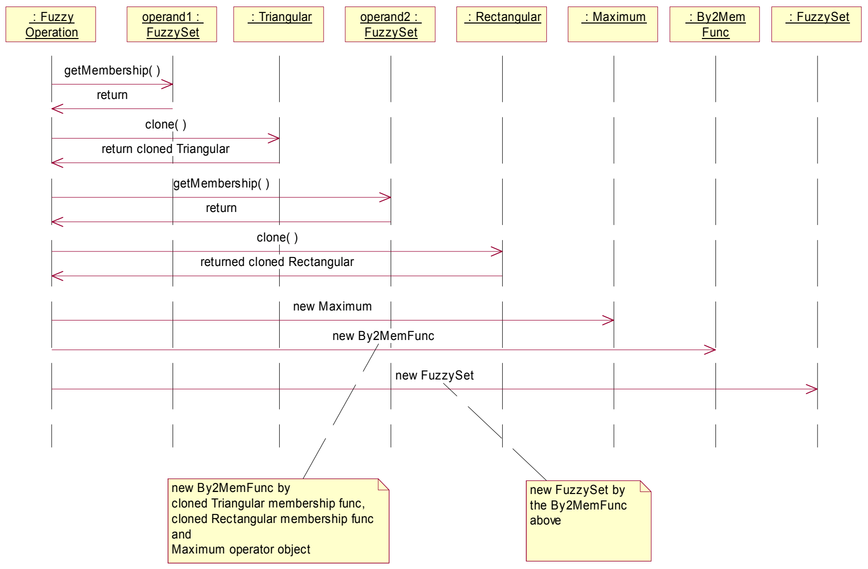

Fig. 3.8. Example: Maximum of fuzzy sets with triangular and rectangular membership function

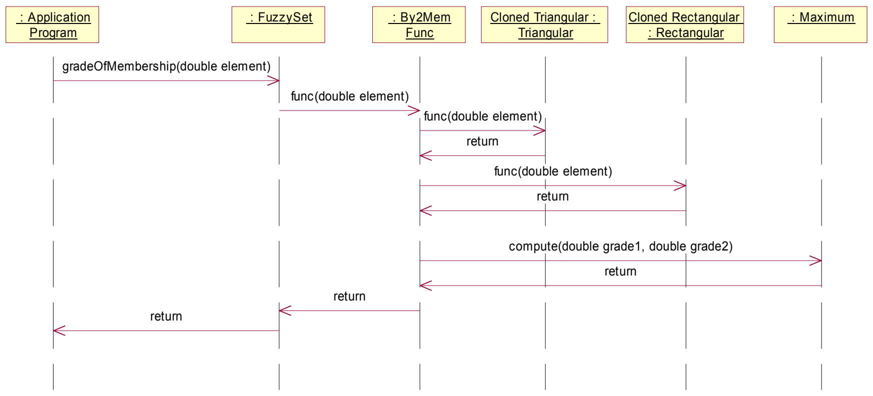

Fig. 3.9. Grade of membership return from the fuzzy set generated by binary operation

When grade of membership is required from the resultant fuzzy set of binary operation. The two-parameter *compute* method in *Operator* object is called instead of the one-parameter *compute* method. As shown in the sequence diagram in Fig. 3.9.

Using the architecture described above, we have implemented a Java class library called **FuzzySys** that facilitates the development of a general class of fuzzy logic systems. In the following Sections, the Java implementation of **FuzzySys** will be described in detail. The concepts we have described can easily be implemented in other object-oriented languages as well.

4 Implementation

In the following, some Java code attached are used to explain the concept described in above Section more clearly.

One of the important techniques to make this approach works, is the concept of copying object. To implement both *FuzzyMembershipFunc* and *FuzzyOperator* class, we need to make it clonable [1], so fuzzy operations can copy the operand fuzzy set's membership function class to create resultant fuzzy set.

For some special membership functions which contain complex data type as attribute, such as *By1MemFunc* and *By2MemFunc*, beside override the *func* method, they also need to override the *clone* method, so the cloned object will not share the same attribute (object) with the original object [1]. Following is part of the *By2MemFunc* code which show the *clone* method and also the *func* method which responsible for a part of operation on Fig. 3.9.

```
public final class By2MemFunc extends FuzzyMembershipFunc {

    protected FuzzyMembershipFunc memFunc1, memFunc2;
    protected FuzzyOperator       operator;

    // Constructor, accessor and modifier define here ...

    public Object clone() {
        By2MemFunc fmem = (By2MemFunc)super.clone();
        fmem.setMembershipFunc1((FuzzyMembershipFunc)memFunc1.clone());
        fmem.setMembershipFunc2((FuzzyMembershipFunc)memFunc2.clone());
        fmem.setOperator( (FuzzyOperator)operator.clone() );
        return fmem;
    }

    public double func(double input) {
        double gradeOfMembership1 = memFunc1.func(input);
        double gradeOfMembership2 = memFunc2.func(input);
        return operator.compute(gradeOfMembership1, gradeOfMembership2);
    }
}
```

For other membership function class without complex data type, it only needs to extends (inherit) from this class and override the *func* method. Following is part of *Rectangular* membership function class Java code.

```
public final class Rectangular extends FuzzyMembershipFunc {
    protected double start, end;

    // Constructor, accessor and modifier define here...

    public double func(double input) {
        if ( input >= start && input <= end )
                return 1;
        else
                return 0;
    }
}
```

Following is part of the Alpha Cut, fuzzy operator code:

```
public final class AlphaCut extends FuzzyOperator {

    protected double alpha;

    // Constructor, accessor and modifier define here ...

    public double compute(double grade) {
        if ( grade < alpha )
                return 0;
        else
                return grade;
    }
}
```

As mention before, fuzzy operations create the resultant fuzzy set by copy the operand(s) fuzzy set's membership function object and new an operator object(s), such as alpha cut, complement ...etc. Following is an example showing how fuzzy operation is done by static methods (See Fig. 3.6 and Fig. 3.8) in *FuzzyOperation* class.

```
public class FuzzyOperation {

    public static FuzzySet complement(FuzzySet op) {
        FuzzyMembershipFunc fmc =
                        (FuzzyMembershipFunc)op.getMembership().clone();
        FuzzyOperator fo = new Complement();
        FuzzyMembershipFunc fmn = new By1MemFunc(fmc, fo);
        return new FuzzySet(op.getDomainMin(), op.getDomainMax(), fmn);
    }

    // Other operations defined here...

}
```

By the implementation above, we do not use array to store sampled order pair to represent fuzzy set; instead we use a function directly and use some attribute to store the characteristics of that function. For example: *start* and *end* in *Rectangular* membership function. Therefore, for system with many fuzzy sets, our approach minimizes memory required. Furthermore, in our implementation, we do not have any loop and the growth of function [7] in our approach is constant. For traditional approach, the growth of functions is in terms of the number of samples. So, our new approach seems to offer advantages in memory space and computation time.

5 Conclusion

The major advantage to our new object-oriented modelling of fuzzy sets is that we avoid problems of over- or under-sampling and the distortion problem in general. Furthermore, because the resultant fuzzy set contains all the fuzzy membership functions and operators involved, the fuzzy set operations are traceable and make application debugging easier. In addition, due to our object-oriented structure, the fuzzy membership functions and operators can be reused and managed in a flexible way. Although increasing cascaded operations will increase the number of computation, this overhead is not large. On the other hand, grades of membership will only be calculated when needed and avoid unnecessary computation of traditional approaches. Therefore in general the performance of our approach will be better.

The traditional method of modelling fuzzy sets with a pair of arrays to store elements and their grades of membership may cause distortion if sampling is not enough. Our object-oriented model solves this problem. We have successfully implemented a Java class library, called **FuzzySys,** which encodes this concept. Our class library not only solves the distortion problem, but also provides user a flexible and efficient way to develop full-scaled fuzzy logic systems.

References

1. Arnold Gosling: The Java Programming Language. Addison Wesley (1997) 77–82
2. Cox, E.: Fuzzy System Handbook. AP Professional (1998) 81–216
3. Flowler Martin. UML Distilled, Applying the Standard Object Modeling Language. Addison Wesley (1997)
4. Grady Booch, James Rumbaugh, Ivar Jacobson: The Unified Modeling Language User Guide. Addison Wesley (1999)
5. Granino A. Korn: Neural Networks and Fuzzy-Logic Control on Personal Computers and Workstation. MIT Press (1991) 315–322
6. R. Lowen: Fuzzy Set Theory: Basic Concepts, Techniques, and Bibliography. Kluwer Academic Publishers (1996) 49–132
7. T.H. Cormen, C.E. Leiserson, R.L. Rivest: Introduction to Algorithms. MIT Press/McGraw-Hill (1990) 23–31
8. Zadeh, Lotfi A.: Fuzzy Set. Information and Control **8** (1965) 338–353
9. Zimmermann, H.J.: Fuzzy Set Theory and Its Applications. Kluwer Academic Publishers (1996)