Designing Fuzzy Relations in Orthogonal Persistence Object-Oriented Database Engines

Miguel-Ángel Sicilia[1], José-Antonio Gutiérrez[2], Elena García[2]

[1] DEI Laboratory, Computer Science Department, Carlos III University
Av. Universidad 30, 28911 Leganés (Madrid), Spain
{msicilia}inf.uc3m.es
[2] Computer Science Department, Alcalá University
Ctra. Barcelona km.33.600, 28871 Alcalá de Henares (Madrid), Spain
{jantonio.gutierrez, elena.garciab}@uah.es

Abstract. Semantic relations between concepts or data are common modeling devices in various knowledge representation approaches. *Fuzzy relations* can be defined as fuzzy subsets of the cartesian product of a number of domains, extending the notion of crisp relation. *Associations* in object-oriented modeling – and more specifically in the *Unified Modeling Language* – can be interpreted as crisp relations on the classifiers they connect, and thus the concept of association can be extended to its fuzzy counterpart by representing a fuzzy relation on the classes involved in the association. In this paper, the *resolution form* of a fuzzy relation is described as a convenient way to represent *fuzzy associations* in object-oriented programming languages, thus enabling an efficient practical representation mechanism for them. Specific cases of fuzzy relations can also be dealt with by adding semantic constraints to fuzzy associations. One of the most interesting cases is that of *similarity relations*, which essentially generalize the notion of object equality to the fuzzy case. Fuzzy relations can be stored in orthogonally persistent object databases by using the described fuzzy association design, as illustrated in this paper with a case study that uses the `db4o` persistence engine.

Keywords. Fuzzy relations, similarity relations, object-oriented databases.

1 Introduction

A number of research groups have investigated the problem of modeling fuzziness (in a broad sense, including imprecision and uncertainty, as defined in (Smets, 1997)) in the context of object-oriented databases (OODB) (De Caluwe, 1998), and some of their results include research implementations on top of commercial systems (Yazici *et al*, 1998). Nonetheless, currently no commercial system is available that supports fuzziness explicitly in its core physical or logical model, and existing database standards regarding object persistence sources – ODMG (Cattell 2000) and JDO (Russell 2001) – do not support neither fuzziness nor any other kind of generalized uncertainty information representation (Klir, 1998) – in their data models.

Nonetheless, imperative OODB application programming interfaces stay very close to the semantics and syntax of the object-oriented programming languages in which they're embedded – see, for example, (Atkinson 1996) – facilitating the construction of research prototypes that extend commercial systems by adding a software layer acting as a *proxy* filter (Gamma 1995) for the underlying non-fuzzy languages.
Relations between concepts are a very common construct in diverse knowledge representation approaches, including modern ontology-description languages (Horrocks, 2002), and as such, they require specific physical representation mechanisms to be efficiently handled by application software. In this work, we describe our approach for the design of fuzzy relations in orthogonal persistent OODBs, evolved from earlier work (Gutiérrez *et al.*, 2002). More specifically, we concentrate on the design of associations between database (or model) entities, and on the specific case of similarity relations. We describe a case study that extends the `db4o` database engine interfaces to include general-purpose *fuzzy associations*, a concept that can be considered an extension of the ODMG relationship construct (Cattell 2000). We also describe a prototype version for the extensions described in this work, tested on the fully functional `db4o` community edition, which can be freely redistributed in non-commercial applications.

2 Fuzzy Relations and Object-Oriented Associations

2.1. Relations and Associations

A *crisp* relation represents the presence or absence of interconnectedness between the elements of two or more sets. This concept is referred to as *association* when applied to object oriented modeling. According to the Unified Modeling Language (UML) (OMG 1999, *Semantics* section, 2-20) – the most widely used object-oriented modeling notation –, an association defines a semantic relationship between *classifiers*[1], and the instances of an association can be considered a set of tuples relating instances of these classifiers, where each tuple value may appear at most once. A binary association may involve one or two fuzzy relations (i.e. the unidirectional and bidirectional cases), although due to the semantic interpretation of associations, they're in many cases considered to convey the same information (i.e. the association between authors and books is interpreted in the same way despite the navigation direction).
Since it's common practice to develop object-oriented software from previously defined UML models, we can consider UML semantics as a model from which associations are implemented in specific object-oriented programming languages, by the process of association design, which essentially consists in the selection of the concrete data structure that better fits the requirements of the association.

[1] A UML term referring to classes and class-like model elements that describe sets of entities.

2.2. Fuzzy Relations

Fuzzy relations are generalizations of the concept of crisp relation in which various degrees of strength of relation are allowed (Klir 1988). A binary fuzzy relation *R* on $X \times Y$ is a fuzzy subset of that cartesian product as denoted in (1).

$$R = \{((x, y), \mu_R(x, y)) \mid (x, y \in X \times Y)\} \quad (1)$$

All the relation concepts can be extended to the n-ary case, where

$$R(X_1, X_2, \ldots, X_n) \subset X_1 \times X_2 \times \ldots \times X_n \quad (2)$$

We'll restrict ourselves to the binary case, since it's the more common case in database applications. Note that even in the recent UML version 1.4, the definition of association relationship is considered to be ill defined (Stevens 2001). Nonetheless, in this work, we'll consider associations as literal tuples between model elements that hold an additional value representing their membership grade to the association. This assumption implies some constraints in the implementation of bidirectional associations, since both association ends should be aware of updates on the other one.

A common representation for fuzzy relations is an *n*-dimensional array (Klir 1988), but this representation does not fit well in the object paradigm, in which a particular object (element of one of the domains in the relation) only is aware of the tuples to which it belongs (the links), and uses them to navigate to other instances. We have extended the association concept to design fuzzy relations attached to classes in a programming language, so that a particular instance has direct links (i.e. '*knows*') to instances associated with it. Access to the entire relation (that is, the union of the individual links of all the instances in the association) is provided as a class responsibility, as will be described later.

3 Design Case Study

The `db4o`[2] object database system is a lightweight OODB engine that provides a seamless Java language binding (it uses reflection run-time capabilities to avoid the need to modify existing classes to make their instances storable) and a novel *Query-By-Example* (QBE) interface based on the results of the SODA[3] – Simple Object Database Access – initiative.

Associations are stored in its Java native form in `db4o`, and therefore, special support for fuzzy associations can be designed by using different standard object oriented design techniques.

[2] Available at <http://www.db4o.com>

[3] Information available at <http://www.odbms.org/soda/>

3.1. Designing Binary Associations

The membership values of the relation must be kept apart from the instances of the classes that participate in the association. A first approach could be that of building *Proxies* for the instances – which will hold a reference to the instance at the other side of the association and the membership grade – and storing them in a standard collection. The main benefit of this approach is simplicity, since only a class called for example *FuzzyLink* (FL from now on) solves the representation problem, and it's enough for the case of association with cardinality one. We used this first approach for comparison purposes with our final design.
A drawback of the FL approach for associations with multiple cardinalities is that the responsibility of preserving relation properties is left to the domain-class designer. This is one of the reasons that pushed us to a second approach in which the *collection* semantics – and not the element semantics – are extended.

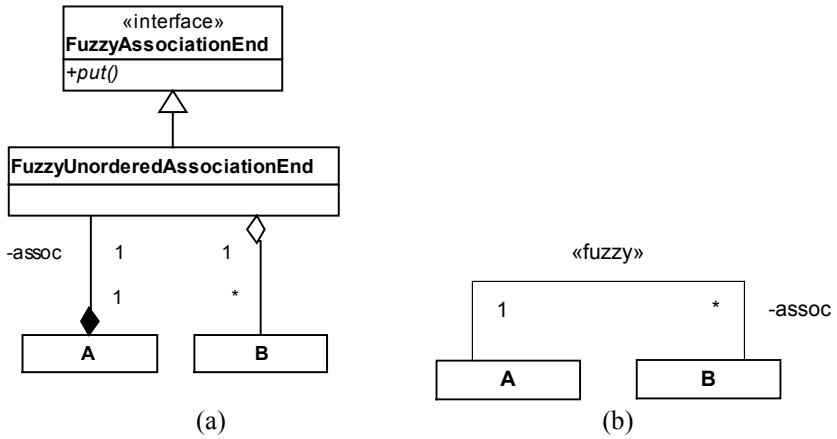

Fig. 1. Unidirectional Binary Association Design

The base of our fuzzy collection framework is a *FuzzyAssociationEnd* (FAE) interface that defines common behavior for all fuzzy associations. Concrete classes implement that interface to provide different flavors of associations. In this work, we'll restrict our discussion to a *FuzzyUnorderedAssociationEnd* (FUAE) class. The class diagram in Fig. 1 shows how a unidirectional fuzzy association[4] (b) from class *A* to class *B* can be designed with our framework (a).
It should be noted that the *put* method can be used both to add and remove objects from the relation – the latter case can be carried out by specifying a zero membership (we have considered in this implementation that zero membership is equivalent to the lack of a link). Since many associations that store different information may exist between the same pair of classes, associations must be named. The class-instance *FUAE* is responsible for maintaining a collection of the associations that are maintained as instances of it (i.e., this behavior is modeled as a class responsibility). These

[4] We have used a <<fuzzy>> UML stereotype to mark fuzzy associations.

different associations are represented by instances a *FuzzyUnorderedAssociation* (*FUA*) class. Therefore, *FUA* instances represent entire generic associations and store the union of the links that belong to it. An example of an association *User-Subject* called '*interested-in*' may be coded in Java as follows:

```
public class Subject{
   public String _name;
   public FuzzyUnorderedAssociationEnd _fasoc;
   public Subject(String name){
       _name = name;
       _fasoc = new FuzzyUnorderedAssociationEnd
                      ("isInterestedIn", false);
    }
   public void registerInterest(Object o, double mu){
     _fasoc.putLink(o, mu);
   }
      public Iterator interestedPeople(){
       return _fasoc.values().iterator();
   }
   /...
}
```

Iterations may be performed on the links with an specialization of the *Iterator* interface that returns the elements wrapped in *FuzzyElement* instances, as shown in the following code fragment.

```
for (Iterator it=xml.interestedPeople();it.hasNext(); ){
  FuzzyElement e = (FuzzyElement) it.next();
  System.out.println(e.getGrade()+(User)e.getObject());
}
```

3.2. Representation details

In our design, links are indexed with their membership values as keys, using sets to hold links with the same value (see Fig. 2). In order to turn feasible this approach, real values representing membership values should be stored with a reasonable precision. Note that more than five decimal numbers is seldom needed in common applications (that is, a tuple with μ_R=0.50001 is hardly distinguishable from another with μ_R =0.5000) in most common applications.

The union of all *FUAE* that belong to the same association is maintained in a *FUA* instance. These instances are stored as a class member of the *FUAE* class (see Fig.3), so that integrity is preserved in insertion and removal of links. This design also enables an easy maintenance of integrity if the association is implemented in both directions.

Using dictionaries with fixed precision-membership values as keys provides performance benefits in common operations on fuzzy sets, like alpha-cuts, outperforming common container classes (bags, sets and lists). The rationale behind this organization is that association traversal would be often done by specifying a minimum membership grade, that is, to obtain an element of the partition of the fuzzy relation.

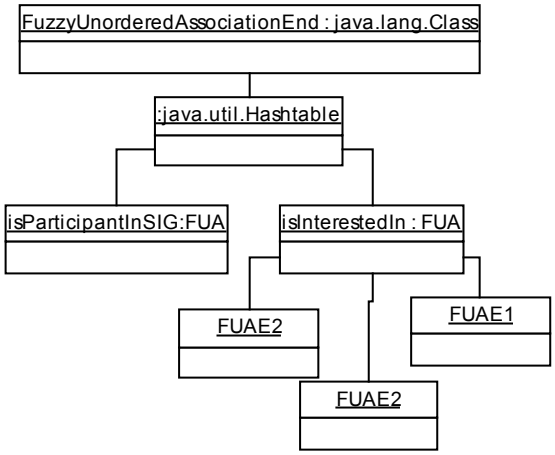

Fig.2. *FUA* instances as the union of *FUAE* ones.

This way, we are representing the relation by its *resolution* form (3).

$$R = \bigcup_{\alpha} \alpha R_{\alpha} \qquad \alpha \in \Lambda_R \tag{3}$$

where Λ_R is the level set of R , R_α denotes an α-cut of the fuzzy relation and αR_α is a fuzzy relation as defined in (4).

$$\mu_{\alpha R_\alpha}(x, y) = \alpha \cdot \mu_{R_\alpha}(x, y) \tag{4}$$

The implementation is an extension of Java's HashMap collection, which essentially substitutes the add behavior with that of a link operation that is sketched as follows:

```
public Object link(Object key, Object value){
   if (key.getClass() == Double.class){
      double mu = ((Double)key).doubleValue();
    // truncates to current precision:
     mu = truncateTo(mu);
    // Get the set of elements with the given mu:
    HashSet elements=(HashSet)this.get( new Double(mu) );
    if ( elements == null ){
           HashSet aux = new HashSet();
           aux.add(value);
          super.put(new Double(mu), aux);
     }else
          elements.add(value);
     }
// Inform the association that a new link has been added:
     if (association !=null)
          association.put(key, this, value);
     return null;
}
```

Figure 3 illustrates our described design by showing a "*is interested in*" relation between the set *U* of the users of a Web site and the set *S* of subjects of the page it serves.

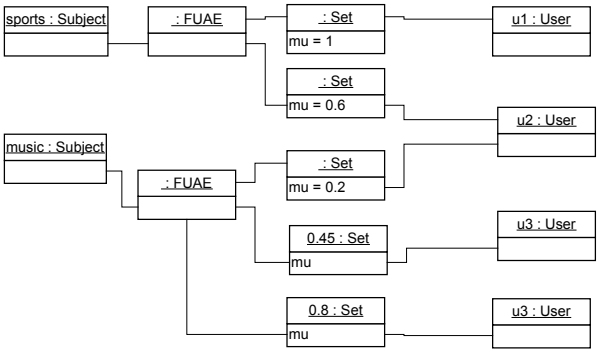

Figure 3. An example "*is interested in*" fuzzy relation.

Some common operations found are obtaining the interest of a user in a specific subject, obtaining the set of subjects a user is interested in and obtaining the set of users which match a specific preference profile (that are interested in a set of subjects). Note that often associations require a specific ordering (i.e. insertion or by an specific attribute). Specializations of the classes of our library are expected to add those behaviors.

3.3. Similarity Relations

Similarity relations can be considered as an extension of the concept of equality, and can be implemented as special reflexive fuzzy associations with added support for similarity semantics. A fuzzy similarity relation is a reflexive, symmetric and transitive fuzzy relation, where reflexivity and symmetry are defined as in (5).

$$\mu_R(x,x) = 1 \qquad (5)$$
$$\mu_R(x,y) = \mu_R(y,x)$$

The transitivity property is usually implemented as *max-mix* transitivity according to the formula in (6).

$$\mu_R(x,y) \le \max_{z \in D}\left[\min(\mu_R(x,z), \mu_R(z,y))\right] \qquad (6)$$

Our approach is that of storing similarity relations defined on a class with crisp boundaries – other approaches define similarity on attribute values for non-crisp classes (Aksoy 1996). In our framework, a *function object* is required at similarity relation construction to specify the concrete transitivity formula (i.e. substituting the inequality with a specific value). An overloaded *put* method version inserts links in the similarity relation without the explicit specification of a membership value. The actual value is derived from the existing ones in the relation, if possible, through the transitive formula.

An overloaded version of `db4o` the query interface (the *ObjectContainer.get()* method) provides access to similarity-enriched QBE queries and gives explicit access to fuzzy associations as well. QBE queries with similarity comparison operators can be constructed by extending the notion of SODA query constraints with fuzzy comparison operators. As the *ObjectContainer* class acts as a factory for its instances, we've extended it through delegation – the class is called *FuzzyObjectContainer*(*FOC*). Basically, when a object whose class holds a similarity relation is passed as a template to method *get* in FOC, not only the instance is retrieved, but also all the similarity related instances. Another version of *get* that takes an additional parameter allows the query to specify a minimum similarity grade for the instances retrieved.

3.4. Performance considerations

The cardinality of the level set of a fuzzy association with precision *m* is bounded by $10^m - 1$, and therefore, the number of sets of links with the same membership value in an association is always below or equal to that number. In a degenerate case, the number of links *L* in the association would be equal to the number of sets, but this is not a common case. In cases in which the number of links is much greater than the number of different membership values, the efficiency of operations that query subsets of the relation increases significantly.

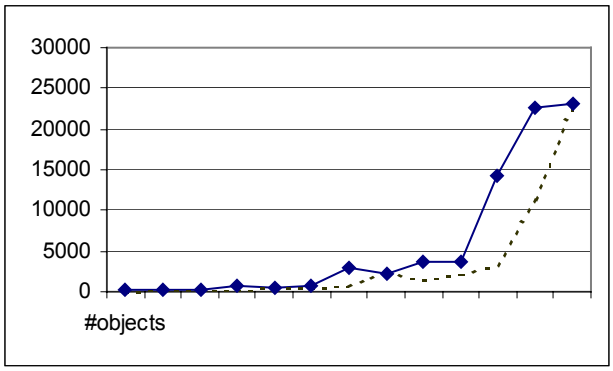

Figure 4. Iteration retrieval time and total number of objects.

Theoretically, the navigational nature of object databases and persistence engines are well suited to operations that only require the physical retrieval of a number of objects, like the retrieval of elements by a membership grade or threshold.

We have carried out several experiments to try to understand the performance behavior of our design. We used a unidirectional fuzzy association from *Subjects* to *Users*, systematically measuring *retrieval* time[5] (understood as the time required to query the association and traverse all its links) for increasing cardinalities of users, subjects and links between them (the membership values for the links were randomly

[5] On a Windows XP machine with 256MB RAM and Pentium IV 1500MHz processor.

generated and the number of links was always above the number of users or subjects). The memory requirements of the Java virtual machine constrained the measures to *total object* (users + subjects) *count* below 10,000 (in the experiment, the number of links was the max(#users, #subjects)). The first finding was that retrieval times increased significantly with total object count above four thousand objects approximately, as showed in Figure 4. We also observed that retrieval time increased slightly less with increases in the number instances of *User* than with increases on *Subject* (which holds the data structure).

The dashed line in Figure 4 summarizes the measures of an alternative implementation using Java `LinkedList` to store the link collection. As showed in that figure, no significant performance overhead is added with our approach from simpler ones. Nonetheless, the retrieval by membership values or by membership threshold (α–cuts) is significantly improved (this is clear from asymptotic analysis, due to the *O(1)* efficiency of search in `HashSet`).

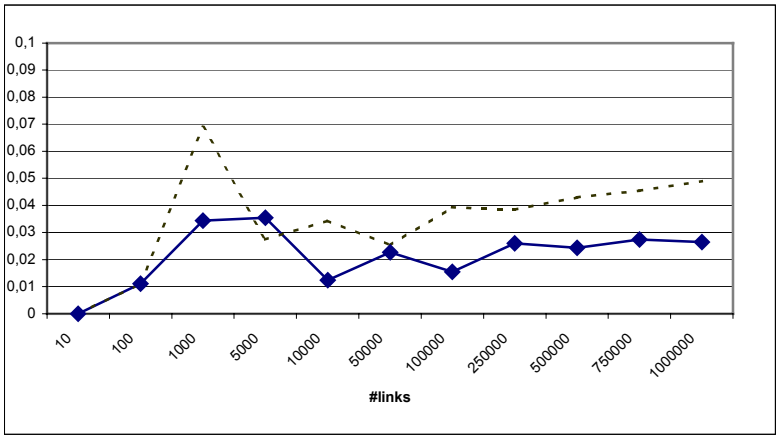

Figure 5. Random α–cut retrieval time increase and number of links.

Figure 5 compares the retrieval time increase with the number of links stored, when retrieving a randomly generated α–cuts. The efficiency decrease is significantly lower in the *FUAE* implementation for large number of links. Sorting the list by membership reduces retrieval time in the medium case, but it still has worse performance than our approach. The `activationDepth` parameter of `db4o` is an important factor in these results. It must be reduced from the default 5 value to 2 or 1 to obtain a significant improvement, since with the default, the entire object graph is always retrieved.

In addition, improvements in retrieval of links by membership were found in comparative experiments with relational-like persistence managers (*MS Access*) for large cardinalities.

4 Conclusions and Future Work

The resolution form of a fuzzy relation is a convenient way of representing fuzzy associations, and subsequently storing them in orthogonal persistence engines. Additional constraints on link insertion semantics can be added to obtain specialized relations like similarity relations. This association design can be stored and retrieved efficiently in object persistence engines by simply obtaining a reference to the object that embodies one of the association ends.

Future work should address a more detailed empirical evaluation on a number of different object persistence systems, and a refinement of the current structure to directly represent α–cuts as objects, which could improve the current approach in retrieving elements by specifying a membership threshold.

Acknowledgements

Thanks to Carl Rosenberg from `db4o` that assisted us in some technical aspects of the storage in the `db4o` database engine.

References

(Aksoy 1996) Aksoy, D. Yazici, A. George, R. : Extending similarity-based fuzzy object-oriented data model. Proceedings of the 1996 ACM symposium on Applied Computing SAC 1996: 542-546

(Atkinson 1996) M.P. Atkinson, L. Daynes, M.J. Jordan, T. Printezis, S. Spence. An Orthogonally Persistent Java, ACM Sigmod Record, Volume 25, Number 4, December 1996.

(Cattell 2000) Cattell, R. G. G. (editor) The Object Data Standard: ODMG 3.0. Morgan Kaufmann Publishers, 2000.

(De Caluwe 1998) De Caluwe, R. (ed.), Fuzzy and Uncertain Object-Oriented Databases: Concepts and Models (Advances in Fuzzy Systems, Applications and Theory, V. 13). 1998.

(Gamma 1995) Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns. Elements of Reusable Object Oriented Design. Addison Wesley (1995)

(Gutiérrez *et al.*, 2002) Gutierrez, J. A., Sicilia, M. A., Garcia: Integrating fuzzy associations and similarity relations in object oriented database systems. In: Proc. Intl. Conf. On Fuzzy Sets Theory and its Applications, FSTA 2002, Liptovský, Slovak Republic (2002) 66-67

(Horrocks, 2002) Horrocks, I.: DAML+OIL: A Reason-able Web Ontology Language. 8th International Conference on Extending Database Technology (EDBT 2002) 2-13

(Klir 1988) Klir, G.J. & Folger, T.A. Fuzzy Sets, Uncertainty and Information, Prentice Hall, Canada Inc., Toronto, 1988.

(Klir, 1998) Klir, G. & Wierman, M. (1998), Uncertainty-Based Information. Elements of Generalized Information Theory, Springer-Verlag, New-York, 1998.

(OMG 1999) Object Management Group, OMG Unified Modeling Language Specification, Version 1.3. June 1999.

(Stevens 2001) Stevens, P., On Associations in the Unified Modelling Language, Proceedings of UML2001, Springer-Verlag, 2001.

(Russell 2001) Russell, C. et al, Java Data Objects (JDO) Version 1.0 proposed final draft, Java Specification Request JSR000012.

(Smets, 1997) Smets, P.: Imperfect information: Imprecision-Uncertainty. In: Motro, A., Smets, P. (eds.) Uncertainty Management in Information Systems: From Needs to Solutions, Kluwer Aca-demic Publishers (1997) 225–254

(Yazici *et al*, 1998) Adnan Yazici, Roy George, Demet Aksoy: Design and Implementation Issues in the Fuzzy Object-Oriented Data Model. Information Sciences 108(1-4): 241-260 (1998)