FUZZY OBJECT COMPARISON AND ITS APPLICATION TO A SELF-ADAPTABLE QUERY MECHANISM

José Nuno OLIVEIRA

INESC Group 2361 / Dep. Informática, Universidade do Minho - 4700 Braga - Portugal

Abstract. This paper describes a generic strategy for "intelligently" querying hierarchical semantic networks whose objects are characterized by fuzzy attributes. The strategy is self-adaptable in the sense that the network will keep structuring itself (as long as the user queries the system) by adding more and more detail to an underlying semi-lattice built on top of a (fuzzy) object comparison partial order.
Technical bounds are imposed on the fuzziness of the system by formally specifying the network and reasoning about it. In the one direction, fuzziness is welcome insofar as it increases the flexibility and discriminating power of the comparison mechanism. In the opposite direction, arbitrary fuzziness may be counterproductive because it will end up destroying the formal properties desirable for such a mechanism.
An experimental illustration of the proposed strategy is presented which re-interprets Prieto-Diáz faceted (fuzzy) scheme for software-component classification and reuse.

1 Introduction

This paper describes the formal basis of the mechanism adopted in the SOUR [1] project for comparing, classifying and retrieving information about large software systems. The unit of information in SOUR is the so-called *abstract object* (AO), a notion which combines the *enumerative* and *faceted* classification [9, 8] schemes in the context of a conventional hierarchical semantic network model extending the popular attributive view of objects. It is because of the shortcomings of this attributive view, for classification purposes [9], that *facets* (or "fuzzy attributes") have been added to the model, in order to cope with features of human reasoning such as *classifying by analogy* and *terminological vagueness*, as reported in the literature [9, 8, 3, 7, 6].

However appealing from an intuitive point of view, the implications of introducing fuzzyness in classification theory have not been formally assessed in the above cited literature. For instance, can the mathematics of Wille [13]'s *concept lattices* be easily extended with fuzzyness?

The main purpose of this paper is to show that *facets* effectively extend "sharp" attributes with respect to classification, and to show how the overall classification scheme can help in designing a self-adaptable query mechanism for user-interface support. The SETS [4] notation is employed throughout the paper for the technical material. Due to its closeness to conventional set-theory notation, it should be easy to follow by informal reader.

2 Motivation

The reuse based approach to software development is today understood as a form of *analogical* problem solving strategy whereby the software engineer attempts to transfer solutions from one system (or problem) to another [11]. The basic question is how to define a quantitative *similarity* relation between software artifacts so as to promote their analogical reuse. Similarity can be described as a relation determined by *comparing* the distinct constituents of two entities. From a quantitative viewpoint, comparison is normally interpreted as a measure of *closeness* in some abstract space.

One of the main motivations of the SOUR project has been to provide for reusability and user-friendliness not in an *ad hoc* way but rather as a beneficial consequence of a formal knowledge subsumption discipline defined beforehand. Although this paper stresses on software reuse, it should be clear that the framework is generally applicable to any kind of computational objects matching the formal specification of the underlying information model.

3 Object Comparison

3.1 General View

Comparison in mathematics is usually expressed by an ordering on the specific domain of reasoning. One may expect that some kind of *partial ordering* can be found on object descriptors (ODs), because not every two descriptors will be linearly comparable. However, should two given objects be incomparable, one might expect that a *least upper-bound* object always exists which is the least abstract object which contains the "common information" of (*i.e.* which generalizes) the given objects. Therefore, the discussion around object descriptor comparison can be expressed in terms of a function

$$comp : OD \times OD \longrightarrow OD \tag{1}$$

which, given two object descriptors (ODs), "computes" their least upper-bound.

Before going into details, one should assert some desirable properties of any candidate $comp$ function, the cornerstone of every comparison algebra:

- the comparison of an OD with itself should yield itself, that is, $comp(x, x) = x$;
- OD-comparison should be commutative, that is, $comp(x, y) = comp(y, x)$ and associative, that is, $comp(x, comp(y, z)) = comp(comp(x, y), z)$;
- there must exist a "most abstract" OD, $\top$, which generalizes any other OD, that is, $comp(\top, x) = comp(x, \top) = \top$.

[1] SOUR is the acronym of EUREKA 379 project "SOftware USe and REuse".

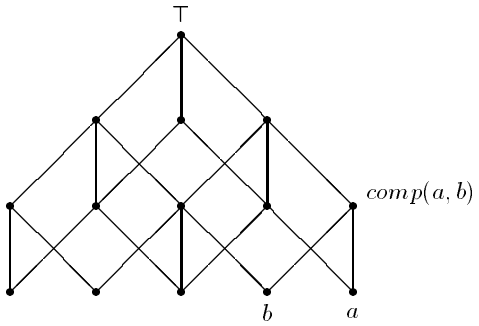

Fig. 1: OD *specialize/generalize* hierarchy.

ODs will thus be assembled in a *specialize/generalize* semi-lattice with $\top$ at the top, see Fig. 1. One will say that x *specializes* y ($\equiv$ y *generalizes* x) wherever $comp(x, y) = y$ holds, and will write $x \leq y$ to denote such a specialization order.

3.2 From Keywords to Facets

We may instantiate the above scheme in the context of specific classification techniques borrowed from classical library theory [9, 8]. For instance, in the *keyword-based approach* (a simple version of enumerative classification), object descriptors are just sets of keywords capturing their most relevant features. Thus we have, in this case:

$$OD = 2^{Keyword}$$

Comparing two ODs x and y means to obtain a "measure" of what they have in common, that is, $comp(x, y) = x \cap y$. So the $\leq OD$-ordering is the inverse of set-inclusion and the most abstract descriptor simply is the empty set of keywords, $\top = \emptyset$ [2].

In the *attributive version* of enumerative classification, OD descriptors are attribute tuples, *i.e.* functions from a domain of *attribute names* to a domain of *attribute values* or *terms* [3]:

$$OD = Name \hookrightarrow Value \tag{2}$$

This description technique encompasses not only record-based descriptors (*e.g.* as in the conventional relational database model) but also *objects* framed into a class-based inheritance system [6]. In this case, what two ODs x and y have in common is captured by selecting every common attribute of both x and y bearing the *same* value [4], *i.e.*:

$$comp(x, y) = \left(\begin{array}{c} a \\ x(a) \end{array} \right)_{a \in dom(x) \cap dom(y) \wedge x(a) = y(a)} \tag{3}$$

The most abstract OD is the totally undefined function, $\top = (\ )$ and the $\leq$-ordering is the inverse of the function definedness ordering:

$$x \leq y \stackrel{\text{def}}{=} dom(y) \subseteq dom(x) \wedge \forall a \in dom(y) : x(a) = y(a)$$

However, the "strictness" of this approach to OD-comparison is well-known (it has been in use for a long time in information retrieval), entailing a too "flat" *specialize/generalize* hierarchy. For instance, comparing descriptor

function: add		*function: insert*
object: client	with descriptor	*object: client*
source: C++		*source: C*

one would obtain

object: client

missing the fact that *C* and *C++* are "close" programming languages and that add and $insert$ may be "synonyms" of each other in the given context.

Clearly, term *synonyms* will produce different descriptors for the same component unless the $x(a) = y(a)$ test in Eq.(3) is relaxed from strict equality to a broader equivalence relation on terms. This problem has been widely discussed in the literature, *cf. e.g.* [9, 8, 3, 7]. To avoid duplicate or ambiguous descriptors, some kind of *vocabulary control* mechanism is on demand. A *term-thesaurus* is required, grouping all synonyms under similar *concepts*. However, term similarity is "fuzzy" in the sense that it cannot simply be decided in the boolean domain of "yes" (full similarity) or "no" (no similarity at all). Similarity should perhaps be "measured" in the range of 0% to 100%. For instance, a vocabulary control expert may decide that term add is 70% similar to sum (arithmetic operator) and only 30% similar to $insert$ (*e.g.* insertion of records in a database, recall the example above).

In general, terminological fuzziness is a natural language problem stemming from two aspects of man-machine communication: *term overloading* ("metaphors") and *vocabulary mismatch* ("idioms") [3]. The former is the use of the same term to mean different concepts (eg. "*add* two numbers" different from "*add* a record to a database"). The latter is the use of a local term (in the local idiom) different from the standard term (eg. *puntatore* instead of *pointer*).

We are thus led to the introduction of *facets*, or "fuzzy attributes", which are discussed next.

4 Facet Comparison

This section provides our own formal fuzzy set interpretation of a *facet term thesaurus* "a la Prieto-Diáz" [9, 8]. As in [1], by a *fuzzy set* we mean a total function $X : U \rightarrow V$, where U is the universe of discourse and V is a completely distributive lattice. For $V = \{0, 1\}$ (the lattice of truth values), we are back to conventional "sharp" sets. For the remainder of this paper we adopt as underlying lattice the usual interval $V = [0, 1]$, whose limits are 0 (no membership) and 1 (100% membership), subject to lattice operations $a \wedge b = min(a, b)$ and $a \vee b = max(a, b)$. A given "level of accuracy" $a \in [0, 1]$ induces a (sharp) subset of U which contains all elements whose membership is guaranteed at (and above) that level of accuracy:

$$X_a = \{x \in U |\ X(x) \geq a\}$$

Of course, $X_0 = U$ and higher accuracies mean smaller sets, *i.e.* $a \leq b$ implies $X_b \subseteq X_a$.

Let $G \subseteq A^2$ be a (sharp) graph on a set A of nodes. A *fuzzy extension* of G, or simply a *fuzzy graph*, is a fuzzy set X over

[2] For this approach to be sufficiently discriminant, $Keyword$ must be more elaborate than a mere collection of keywords. Normally, classification power is increased by structuring the keyword domain on the basis of a hierarchical *taxonomy* of subjects (*e.g.* the Dewey Decimal Hierarchy), as explained in [9, 8] and specified in [6].

[3] Read $A \hookrightarrow B$ informally as "arrays of B's indexed by A's".

[4] As usual in many notations, $dom(x)$ denotes the domain of definition of function x, see *e.g.* [2, 4].

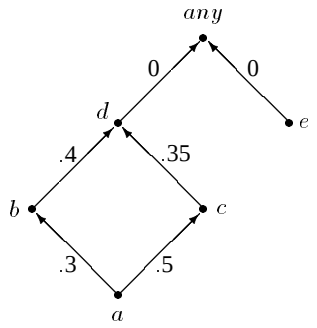

Fig. 2: A fuzzy term graph.

G, that is, a function $X : G \to [0, 1]$. We will write $x \to_a y$ to mean an arc from x to y with membership (accuracy) a, that is, $X(\langle x, y\rangle) = a$.

A *term graph* is a fuzzy graph X on an acyclic graph G whose nodes are terms and whose arcs $x \to_a y$ express the fact that x is a synonym of y at accuracy a. The fact that x is 100% synonym of itself is assumed although not recorded in the graph (it would otherwise be cyclic). Transitive synonyms are guaranteed at the lowest accuracy involved, that is, $x \to_a y$ and $y \to_b z$ implies $x \to_{a \wedge b} z$.

Now let us see under what conditions can a *term graph* X support facet comparison. In a way, we want this graph to represent a fuzzy equivalence relation on terms which should introduce more and more synonyms as membership level decreases. At level 0 we become "absolutely blind" and every term is a synonym of any other. At level 1, we are as strict as possible, only equating 100% synonyms (thus including the identity relation). In a way, we want our term graph to behave like a "fuzzy extension" of the Fischer-Galler "forest model" of equivalence relations [2], storing one equivalence class per maximal element in the graph [5]. Consider, for instance, the term graph of Fig. 2. At any level $l \geq .5$ only $a \to_l c$ holds and thus we are very close to the identity relation (only a and c are synonyms). Further down, within interval $[.35, .4]$, there are two more synonyms: b and d. At $0 < l \leq .35$ equivalence classes $\{a, c\}$ and $\{b, d\}$ collapse with each other: $\{\{a, b, c, d\}, \{e\}\}$. Finally, at $l = 0$ all $\{a, b, c, d, e\}$ are packed into the same equivalence class under the most generic term of all, any (Fig. 2).

Note that not every arbitrary term graph can support the above mechanism. The overall constraint to impose on term fuzzyness is that, at any level $a \in [0, 1]$, the transitive predecessors of all maximal elements of X_a are disjoint sets (the desired "equivalence classes"). Formally, let $\downarrow_a x$ denote the set of all transitive predecessors of node x in X_a, for $a \in [0, 1]$, and let M_a be the set of maximal elements of X_a. The following property must hold:

$$\forall a \in [0, 1], m \neq m' \in M_a : \downarrow_a m \cap \downarrow_a m' = \emptyset$$

As a counter-example, suppose that in Fig. 2 we had $c \to_{.2} d$. Which of c or d is the root term of a for the membership interval $]0.2, .3]$? Because we want to equate everything at level 0 and $X_0 = G$, it is easy to see that G must be universally upper-bounded by a single, most generic term (*cf.* any).

In this way, facet comparison works by "fuzzy unification". At a given membership level, every facet is unified with its root (maximal term) and then comparison takes place. It is interesting to note that, at level 0, we don't get the trivial ordering on facet descriptors. Since every facet is unified with any, it is as if $Value \cong \{any\}$ in Eq.(2). But, in the SETS calculus we have $\{any\} \cong 1$ [6] and law

$$A \hookrightarrow 1 \cong 2^A$$

cf. [4]. So at level 0 it is as if we had

$$OD \cong 2^{Name}$$

that is, the OD-ordering becomes the inverse of the power-set ordering on $Names$ [7]. On the whole, the OD specialize/generalize semi-lattice becomes fuzzy, a topic we have to omit here for space economy (see [5] for details).

5 About the SOUR Information Model

The SOUR repository is a collection of abstract objects (AOs). Each AO, which is internally identified by a unique identifier (Id), is split into two main parts [8]: the object descriptor for classification purposes (OD) and a collection of links to other objects ($Links$). Every OD is in turn a pair of descriptors, one the enumerative descriptor (ED) and the other the faceted descriptor (FD). Altogether, the AO repository is a hierarchical semantic network formally specified by the following system of definitions:

$$\begin{cases} Repository & = & Id \hookrightarrow AO \\ AO & = & OD \times Links \\ OD & = & ED \times FD \\ Links & = & LnkName \hookrightarrow 2^{Id} \\ ED & = & AttName \hookrightarrow AttValue \\ FD & = & FacName \hookrightarrow Term \end{cases} \quad (4)$$

where $Id, LnkName, \ldots, Term$ are primitive types. Terms are handled in a (fuzzy) term-graph as explained in section 4. The AO-specialize/generalize semi-lattice (Fig. 1) is another component of the system, internally maintained in the form of an Id-graph. At initialization time, this graph contains by default only the topmost object $\top$ (Fig. 1). Every time a new AO is entered in the repository, its Id is saved in the appropriate place of this graph by resorting to the $comp\ OD$-comparison function Eq.(1) in a way that ensures that the graph always keeps the minimal, finite sub-semi-lattice of the (denumerably infinite) semi-lattice of all possible AOs [9].

6 Impact on the Query Mechanism and User Interface

Every time a query is fired in the system, the user specifies the desired fuzzyness level. At level 1, only 100% synonyms are considered on the faceted classification side. Should the

[5] Every such element is termed a "root" in the Fischer-Galler terminology or a "concept" in term-thesaurus terminology.

[6] The $\cong$ symbol denotes set-isomorphism, *cf.* [4].

[7] Intuitively, we are "blind" enough to be able only to record which facet names have been assigned *some* value!

[8] Due to lack of space, the actual structure of AOs is here made simpler, omitting the class inheritance mechanism and "compoundness" of AOs. See *e.g.* [10, 12] for details of the actual SOUR system prototype.

[9] Note that, because two different AOs can share the same OD, the actual AO-ordering induced by the OD-ordering is a pre-order and not a partial order. In SOUR, an equivalence table of Ids is maintained for this purpose [10]. The prospect of encompassing the whole AO structure in the comparison is described in [5].

enumerative component not be present, at fuzzyness level 0 the query would select almost all the objects currently stored in the repository (see section 4).

At the user interface level, the repository can be browsed in two main directions via a generic graph-browser tool (the *Result Manager*): not only can the user navigate in the AO semantic network, across user-defined links — *cf.* $Links$ in (4) — but also up and down the AO-comparison graph, across specialize/generalize links. At any time, the user can select objects, compare them and save the result of the comparison. This is particularly useful for top-down search because every object is positioned in this graph above all its known specializations, that is to say, AOs can be regarded as "search templates". For instance, the collection of AOs below the AO which contains a single attribute *source* whose value is *C++* coincides in fact with the result of the query

```
SELECT ALL ...  WHERE SOURCE = C++
```

So the paradigm is to regard a "query as an abstract object", extending the "query as a concept" paradigm of [3] [10]. Complex AND and OR-queries can be decomposed in the intervening AOs. These can be saved in the specialize/generalize graph and their specializations adequately collected for computing the result of the query. Because this process is not visible to the user, the overall mechanism is self-adaptable in so far as more and more detail is added, throughout interaction, into the areas most focussed by the user attention.

Query optimization is another benefit of maintaining the AO specialize/generalize graph. For instance, by inspecting this graph it can be deduced that the following query simplifications,

q AND $q' \equiv q$

q OR $q' \equiv q'$

can take place wherever $q \leq q'$, for q, q' two AO-reducible subqueries.

7 Conclusions and Future Work

The SOUR information retrieval mechanism has the expressive power of what may be termed a "fuzzy SQL processor" equipped with a highly interactive user interface [12].

Due to performance limitations, AO-comparison has been implemented in the current SOUR prototype (running on WINDOWS [10]) in a less complex way than described in this paper and omitting some other aspects of the "full" comparison mechanism investigated in [5], which has some theoretical difficulties concerning arbitrary links and sub-objects. The "query as an abstract object" paradigm is still subject of ongoing research due to the need to consider arbitrary queries and not only the ones which can be easily mapped onto AO counterparts.

The current approach to AO-comparison was inspired by [13]'s lattice-theoretical illustration. The extension of the AO *specialize/generalize* hierarchy to a full complete lattice is suggested in section 4 but needs further research.

Acknowledgements

The author wishes to thank Dr. Henrik Larsen of Roskilde University, Denmark, for his interest in this paper, and thank all his colleagues in the SOUR consortium (INESC, SYSTENA, SSS and OIS RICERCA) who contributed to the many discussions along the project's lifetime.

[10] But note that queries involving relational operators need further elaboration of the paradigm.

References

[1] J. A. Goguen. Concept representation in natural and artificial languages: Axioms, extensions and applications for fuzzy sets. In *Fuzzy Sets*, pages 67–115, 1974.

[2] C. B. Jones. *Software Development — A Rigorous Approach*. Series in Computer Science. Prentice-Hall International, 1980. C. A. R. Hoare.

[3] H. L. Larsen and R. R. Yager. The use of fuzzy relational thesauri for classificatory problem in information retrieval and expert systems. *IEEE Trans. Syst.,Man, Cybern, SMC*, 23(1):31–41, 1992.

[4] J. N. Oliveira. Software reification using the sets calculus. In *Proc. of the BCS FACS 5th Refinement Workshop, Theory and Practice of Formal Software Development, London, UK*, pages 140–171. Springer-Verlag, 8–10 January 1992.

[5] J. N. Oliveira. Fuzzy comparison of recursive objects: A case study. Technical report, DI/INESC, 1995. (in preparation).

[6] J. N. Oliveira and A. M. Cruz. Formal calculi applied to software component knowledge elicitation. Technical Report C19-WP2D, DI/INESC, December 1993. IMI Project *C.1.9. Sviluppo di Metodologie, Sistemi e Servizi Innovativi in Rete*.

[7] E. Ostertag, R. Prieto-Diáz, and C. Braun. Computing similarity in a reuse library system: An ai-based approach. *ACM Transactions on Software Engineering and Methodology*, 1(3):205–228, July 1992.

[8] R. Prieto-Diáz. Implementing faceted classification for software reuse. *CACM*, 34(5):89–97, May 1991.

[9] R. Prieto-Diáz and P. Freeman. Classifying software for reusability. *IEEE Software*, 4(1):6–16, January 1987.

[10] Syntax Sistemi Software. Comparator and modifer—functional specification and architecture. Technical report, SOUR Project, 1993. Ver.1.4, © SSS, Via Fanelli 206-16, Bari, Italy.

[11] G. Spanoudakis and P. Constantopoulos. Similarity for analogical software reuse: A conceptual modelling approach. Technical Report 92-08, NATURE Report Series, 1992. ESPRIT Project 6353.

[12] Systena and Syntax Sistemi Software. Integrated sour software system—demo session manual. Technical report, SOUR Project, 1994. Ver.1.2, © Systena & SSS, Via Zanardelli 34, Rome & Via Fanelli 206-16, Bari, Italy.

[13] R. Wille. Restructuring lattice theory: An approach based on hierarchies of concepts. In I. Rival, editor, *Ordered Sets*, pages 445–470. D. Reidel Pub. Company, 1982.