Applying Document Retrieval Techniques in Software Engineering Environments

Andreas Henrich

Praktische Informatik, Fakultät Sozial- und Wirtschaftswissenschaften,
Otto-Friedrich-Universität Bamberg, D-96045 Bamberg, Germany,
e-mail: andreas.henrich@sowi.uni-bamberg.de

Abstract. In previous papers we have proposed the integration of content-based document retrieval facilities into the repository underlying a software engineering environment. In the present paper we describe the application of these facilities in typical application areas, such as searching reusable components or identifying project dependencies. Furthermore we describe the integration of the facilities into a concrete software engineering environment.

1 Introduction

Most modern software engineering environments utilize something which might be called an *object base* as the repository for the data used by the tools of the environment. The main objective of the repository is data integration, i.e. the sharing of data/documents between multiple tools. As a side effect a valuable information pool accrues. This information pool comprises documents of various types from various projects. Unfortunately repositories usually provide only limited query facilities to exploit this information pool. For example the ISO repository standard PCTE [6] provides only navigational access to the data and other repositories based on relational database technology provide an SQL query language. However, various situations can arise where more powerful retrieval facilities would be useful. Two typical scenarios are as follows:

- The sales department states the request for a new information system supporting their field service representatives, and the IT department has to draw up a prestudy for the intended project. In this case it would be desirable, to search the repository for relevant and/or reusable documents from other projects. To this end, a search by content would be useful. For example, a short description of the new project could be used to search for documents with similar content in the repository.
- To avoid possible inconsistencies the quality assurance group of a company wants to search for redundant class definitions in the repository. To this end, it would be useful, to determine the most similar pairs of classes, which are no relatives of each other in the inheritance hierarchy. The similarity can for example be defined on the content of the class definitions, on the associated attributes and on the signatures of the associated methods.

In practice information needs like those mentioned in the first example above are often addressed using pattern matching facilities like the `grep` command in UNIX. Although these pattern matching facilities are useful in many situations they have certain drawbacks: (1) They do not consider different forms of a word or synonyms, (2) they do not allow for the definition of a content-based similarity measure between documents, and (3) they do not allow for a ranking of the documents with respect to their relevance for the information need.

To overcome these problems content-based search techniques have been developed in the field of information retrieval. The techniques developed allow for example to search for the documents most similar to a given query text, where similarity is defined content-based. The abstraction from the concrete wording of a document is achieved for example using stemming techniques and techniques to exploit the correlation between different terms.

In [4] we have presented P-OQL, an OQL-oriented query language for PCTE combining the usual declarative access ("`select ... from ... where`") with content-based retrieval facilities (cf. section 2). Furthermore in [3] and [5] we have proposed special access structures for the efficient evaluation of corresponding queries. The main contribution of the present paper is to describe the application of these retrieval facilities in a broad variety of situations (cf. section 3) and to sketch their integration in a software engineering environment (cf. section 4). In particular we will describe the various opportunities to combine similarity-based queries with additional conditions and to define new application specific similarity measures dynamically.

2 The Repository

PCTE (*Portable Common Tool Environment*) is the ISO and ECMA standard for a public tool interface (PTI) for an open repository [6]. As one of its major components PCTE contains a structurally object-oriented object management system (OMS). The data model of the PCTE OMS can be seen as an extension of the binary Entity-Relationship Model. The object base contains objects and relationships. Relationships are normally bidirectional. Each relationship is realized by a pair of directed links, which are reverse links of each other. PCTE offers five different link categories: *composition* (defining the destination object as a component of the origin object), *existence* (keeping the destination object in existence), *reference* (assuring referential integrity and representing a property of the origin object), *implicit* (assuring referential integrity) and *designation* (without referential integrity).

Let us consider the example schema for OOA-diagrams in figure 1. As usual, object types are given in rectangles, attribute types are given in ovals, and link types are indicated by arrows. A double arrowhead at the end of a link indicates that the link has cardinality *many*. Links with cardinality *many* must have a key attribute. In the example the numeric attribute *no* is used for this purpose. A *"C"* or *"R"* in the triangles at the center of the line representing a pair of links, indicates that the link has category *composition* or *reference*.

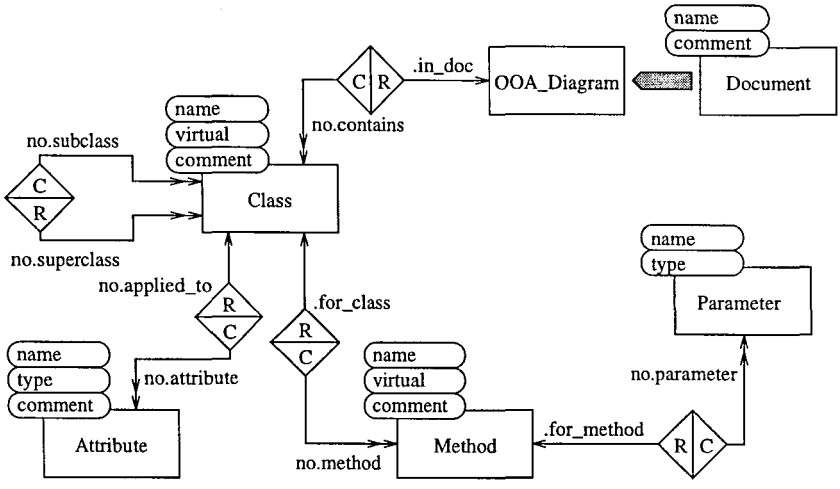

Fig. 1. Simplified example schema for OOA-diagrams

Finally the schema contains a subtype relationship between the object types *Document* and *OOA_Diagram* which is indicated by a broad shaded arrow.

P-OQL [2] is an OQL-oriented query language for PCTE. Assume that we search for the name and the applicable methods of all classes in the OOA-diagram named *"Clearing"*. The following query in P-OQL yields this information:

```
select C:name, normalize C:[_.superclass]*/_.method/->name
  from D in OOA_Diagram, C in (D:_.contains/->.)
  where D:name = "Clearing"
```

In the **from-clause** two base sets are defined: Base set `D` addressing all OOA-diagrams and base set `C` addressing all objects which can be reached from the actual object of base set `D` via a path matching the regular path expression "`D:_.contains/->.`". In this path expression the prefix "`D:`" addresses the actual element of base set `D`. "`_.contains`" means that exactly one link of type *contains* must be traversed. The underscore ("`_`") is used as a wildcard for numerical key attributes. "`/->`" is used to address the destination object of the path. In addition "`->`" can be used to address the last link of the path. The "`.`" at the end of the regular path expression means that the object under concern is addressed. It is also possible, to address an attribute or a tuple of values.

In the **where-clause** of our example query the considered combinations of OOA-diagrams and classes are restricted to those for which the *name* attribute of the OOA-diagram has the value *"Clearing"*.

The **select-clause** states that a multiset of pairs is requested. Each pair consists of the name of the class and a set containing the names of the methods which can be applied to instances of the class. Since methods defined for superclasses of the class under concern can also be applied to in-

stances of the class, we have to address these methods as well. To this end, the regular path expression "`C:[_.superclass]*/_.method/->name`" is used. "`[_.superclass]*`" defines that zero or more links matching the link definition "`_.superclass`" have to be traversed. Alternatively P-OQL knows the iteration facilities "`[` *path_definition* `]+`" to indicate that a path matching *path_definition* has to be traversed at least once and "`[` *path_definition* `]`" to indicate that a path matching *path_definition* is optional. Summarizing, the regular path expression "`C:[_.superclass]*/_.method/->name`" determines a multiset containing the names of all methods which can be reached from the class object under concern traversing an arbitrary number of *superclass* links and exactly one link of type *method*. This multiset is transformed into a set using the unary operator `normalize`.

In addition to the link definitions used in the above example P-OQL allows the specification of a set of link categories with the meaning that all links having one of the given categories fulfill this link definition. E.g. the expression "`[{c, e}]+/->.`" addresses all objects which can be reached via a path consisting only of links with category *composition* or *existence*. Finally an additional `shield`-option can be used to prevent traversing a link of a given type or a link with a destination object of a given type. For example the link definition "`{c shield subclass}`" allows to traverse arbitrary *composition* links except those of type *subclass*.

As mentioned in the introduction one main feature of P-OQL is the integrated **document retrieval facilities**. These facilities implement the vector space model [9,7]. This model assumes that an available term set (vocabulary) is used to identify both, maintained documents and information requests. Queries and documents are represented as *term vectors* of the form $D_i = (a_{i1}, a_{i2}, \ldots, a_{it})$ and $Q_j = (q_{j1}, q_{j2}, \ldots, q_{jt})$ where t is the number of the terms in the term set and where the coefficients a_{ik} and q_{jk} represent the relevance of document D_i or query Q_j, respectively, with respect to term k. In the literature various term-weighting formulas have been proposed to calculate the a_{ik} and q_{jk}. In P-OQL we employ the formulas presented in [8] which have been proved to be competitive e.g. in [1].

To integrate these formulas into P-OQL, we use three operators. First there are the unary operators `D_vector` and `Q_vector` which determine a document or query description vector, respectively. Roughly spoken these operators calculate the vector components for the terms based (1) on the term frequency of the term in the document under concern and (2) on the inverse collection frequency which is high for terms occurring in only a few documents and low for terms occurring in many documents (see [4] for the details). The third operator is the binary operator `sim` calculating the conventional vector product of a document and a query description vector. This yields high values for "similar" documents and low values for "unrelated" documents.

The operator `D_vector` can be applied to *document definitions*. A *document definition* can e.g. be (1) a string attribute, (2) an object (in this case all string attributes of the object are considered), or (3) a collection (set, multiset, list, or

structure) of the before-mentioned entities. Such a collection can e.g. be defined by a regular path expression.

If we are e.g. looking for non-virtual class definitions similar to a given textual class description, it would be appropriate to consider each *Class* object together with the associated attribute and method definitions as a *"class definition document"*. This is done in the following query applying `D_vector` to a set of objects determined by the path expression "`[{c shield subclass}]*/->.`". The query determines the 10 classes most similar to the textual class description:

```
head[10]
  sort-
    (select (Q_vector "<textual class description>"
             sim D_vector [{c shield subclass}]*/->.),
             name
       from Class
       where virtual = FALSE)
```

In this query the query text is given as a string constant. The `Q_vector` operator is applied to this query text to create the query description vector. The document description vectors are determined applying the `D_vector` operator to the *"class definition documents"*.

The select-statement yields a multiset of pairs, where the first component contains the similarity value for the actual *"document"* and the second component contains the value of the *name* attribute of the *"document"*. The `sort-` operator is applied to the result of the select-statement yielding a list with the pairs sorted descendingly according to their similarity values. Then the `head` operator is used to extract the 10 elements at the beginning of this list.

3 Application Examples

3.1 Addressing Documents of Various Types

In software development projects situations are common, where someone searches for documents dealing with a specific topic. Assume for example that a new project is set up for a specific problem. In this case the project leader will be interested in all documents created by previous projects which address this problem area. In this situation the type of these documents is of minor importance. Fortunately the generic path definitions based on link categories allow for such type independent queries in P-OQL. For a given query text — for example a short description of the new project might be used for this purpose — the following query searches the 25 most similar documents in the repository:

```
head[25]
  sort-
    (select (Q_vector "query text" sim D_vector [{c}]*/->.),
            .,
            type of .
       from Document^)
```

In the from-clause of this query "`Document^`" defines that all objects are addressed which are either of type *Document* or of an arbitrary subtype. In the select-clause the result items are defined as triples. In the first component the similarity value is calculated. For the calculation of the description vector for the document we consider all string attributes of objects which can be reached traversing zero or more *composition* links. In the second component the dot defines that an object reference of the object under concern is required. This object reference can be used to apply other PCTE operations and it can be passed to a tool which deals with objects/documents of the corresponding type. Finally in the third component the type of the document under concern is determined.

3.2 Content-Based Retrieval with Additional Conditions

Besides the broad queries described in the previous section, precisely restricted queries are as well needed during software development. One usual requirement in this respect is to combine a content-based query with additional conditions. For example one might be interested only in documents created by a certain person or after a certain date, or one might be interested in method definitions with a special signature. Due to the homogeneous integration of the document retrieval facilities into P-OQL the query language provides unlimited freedom in this respect. As an example, we can search for the classes most similar to a given query text and in addition demand that the classes should not be virtual and have an attribute named *"state"*:

```
head[25]
  sort-
    (select (Q_vector "query text"
             sim D_vector [{c shield subclass}]*/->.),
            .
       from Class
       where virtual = false
         and exists A in _.attribute/->. with A:name =[1] "state")
```

In the exists quantifier all attributes of the actual class are addressed and it is demanded that at least one name of an attribute has an edit distance to *"state"* of at most 1.

Another opportunity is to address the structure of the complex objects under concern. For example we can search for the methods most similar to a given query text which have at least two parameters of type *"float"* and one parameter of type *"integer"* (note that constant multisets are defined as `M{ ... }` in P-OQL):

```
head[25]
  sort-
    (select (Q_vector "query text" sim D_vector [{c}]*/->.), .
       from Method
       where M{"float", "float", "integer"} <= _.parameter/->type)
```

3.3 User-Defined Similarity Measures

One obvious drawback of the document retrieval facilities integrated in P-OQL is that they address only the content-based similarity of the textual parts of the documents. On the other hand, we can utilize the expressive power of P-OQL to define new similarity measures. Assume for example that we are interested in methods. Obviously there are different ways to define the similarity of methods. One opportunity would be to address the signatures, another opportunity might be to address the control structures and last but not least the similarity can be defined on the text written in the comment attribute. Depending on the concrete situation it might be useful to combine multiple similarity measures. As an example the following query combines the content-based similarity of the comment attributes and the similarity of the signatures, where the desired signature consists of two float parameters and one integer parameter:

```
head[25]
  sort-
    (select ((Q_vector "query text" sim D_vector comment)
             * count (M{"float", "float", "integer"}
                      intersect _.parameter/->type)),
            .
      from Method)
```

In this query the cardinality of the intersection of the desired signature and the actual signature is multiplied by the content-based similarity of a given query text and the comment attribute. This way we yield a symmetric combined measure where a higher similarity with respect to one basic similarity measure can compensate a lower similarity with respect to the other similarity measure. It is also possible to combine similarity measures in a hierarchical way. Assume for example that the signature-based similarity should be the dominant one. Then the similarity for the comment attribute should be considered only for methods which have the same signature-based similarity value. Since "`Q_vector "query text" sim D_vector comment`" yields results in the interval $[0,1]$ and "`count (M{"float", "float", "integer"} intersect _.parameter/->type)`" yields values out of $\{0,1,2,3\}$ this can be achieved by the following query:

```
head[25]
 sort-
  (select (count (M{"float", "float", "integer"}
                  intersect _.parameter/->type)
           + 0.5 * (Q_vector "query text" sim D_vector comment)),
          .
    from Method)
```

3.4 Selecting Similar Pairs

As mentioned in the introduction, the information retrieval facilities can also be used to search for redundant documents indicating double work and potential

inconsistencies. Assume for example that we are trying to avoid unrelated classes implementing similar things. In this case we could search for pairs of classes with a high similarity, which are not related to each other in the inheritance graph:

```
head[25]
  sort-
    (select (D_vector A:[{c shield subclass}]/->.
             sim D_vector B:[{c shield subclass}]/->.),
            A:.,
            B:.
       from A in Class, B in Class
       where A:. < B:.
         and A:. !in B:[_.subclass]+/->.
         and B:. !in A:[_.subclass]+/->.)
```

The first condition in the qualification-clause of this query assures that (1) no pair containing the same class in both components is returned and that (2) the pairs are not reported twice — as (a, b) and as (b, a). The second and the third condition assure that neither a is a subclass of b nor b is a subclass of a.

Summarizing, the above query yields the 25 most similar pairs of classes which are not related in the inheritance graph. For these pairs an expert might now consider if the classes should be in some inheritance relation or if they are really independent.

4 Integrating the Retrieval Facilities into a CASE Tool

The major aim of PCTE and P-OQL is to support tool developers. Tool developers should use the high level functionality of PCTE and P-OQL via the application programming interface (API) to develop new tools with a reasonable effort. Nevertheless, it seems to be worthwhile, to integrate a special tool into a software engineering environment which enables the user of the software engineering environment to state similarity searches directly.

Figure 2 shows the user interface of our tool. The main window comprises three subwindows.

- In the subwindow *"Addressed Entity"* the *root objects* of the addressed documents are defined. To this end, an object type can be selected from a menu. This menu also shows the inheritance relations between the object types. In addition the user can define that objects of subtypes should be addressed as well, and he can state a selection predicate in the usual P-OQL syntax.
- In the subwindow *"Address As Document"* the attributes or objects containing the content of each document can be defined. The user can either select one of the three standard opportunities or he can state a regular path expression in the P-OQL syntax.
- In the subwindow *"Query"* the information need can be defined either by a query text, or by selecting and weighting terms from the vocabulary, or by a regular expression.

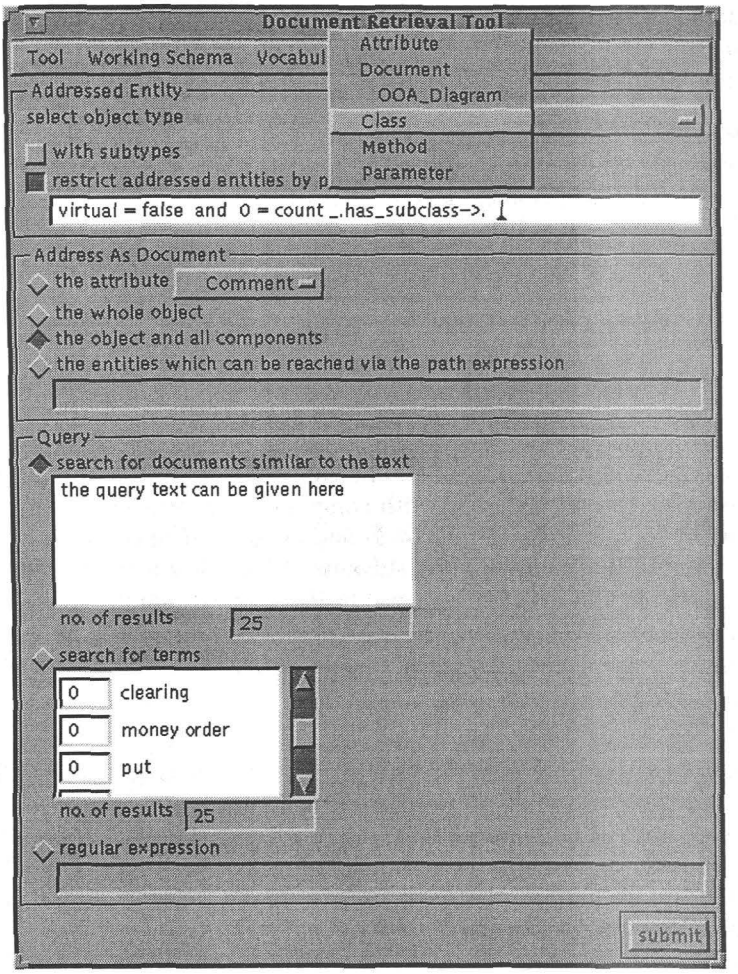

Fig. 2. An end-user interface for the document retrieval facilities

When the *"submit"*-button is pushed, a P-OQL query is created based on the selections of the user. In addition the tool searches for an attribute *name* or *Name* applied to the selected type for the object representing the document. If such an attribute exists, it is integrated into the query result to enhance the readability of the result. Furthermore the type of the object is addressed using the `type of` operator.

In case the option *"search for terms"* is chosen in the subwindow *"Query"*, the application of the `Q_vector` operator is replaced by an explicitly given query vector (which is stated in an abbreviating notation). If the option *"regular expression"* is chosen in the subwindow *"Query"*, the query is transformed into a query applying the pattern matching facilities of P-OQL described in [4].

Once a query is submitted, the next problem is the presentation of the query result. To this end, we employ a standard result viewer. In this viewer all values except of link references and object references are presented in a formatted way. Link and object references are represented by buttons. If a button representing an object is pushed, the viewer creates a list with all objects in the repository containing the object under concern as a component. In this list the objects are sorted with respect to the length of the paths connecting the object in the list and the object under concern. Then the tool registration module of the software engineering environment is asked for each element in the list if there are tools registered for the corresponding object type. If exactly one tool exists for all elements in the list, this tool is started. If more than one tool exists, a tool selection menu is displayed where the user can choose the tool he wants to start in order to work with the selected object. If no tool exists, a standard object browser is opened. This browser displays the attributes and links of the object under concern and allows to navigate across the links. While navigating with the browser, the browser always asks the tool registration module of the software engineering environment for the tools available for the actual object and the user can start these tools from the menu bar.

Obviously similar interfaces can be built for other types of queries. Furthermore the presented tool allows to state queries in plain P-OQL. Since queries can be saved, this allows experienced users to compose queries for different purposes, which can be utilized by other users.

References

1. D. Harman. Overview of the second text retrieval conference (TREC-2). *Information Processing and Management*, 31(3):271–289, 1995.
2. A. Henrich. P-OQL: an OQL-oriented query language for PCTE. In *Proc. 7th Conf. on Software Engineering Environments (SEE '95)*, pages 48–60, Noordwijkerhout, Netherlands, 1995. IEEE Computer Society Press.
3. A. Henrich. Adapting a spatial access structure for document representations in vector space. In *Proc. 5th Intl. Conf. on Information and Knowledge Management*, pages 19–26, Rockville, Md., USA, 1996.
4. A. Henrich. Document retrieval facilities for repository-based system development environments. In *Proc. 19th Annual Intl. ACM SIGIR Conf. on Research and Development in Information Retrieval*, pages 101–109, Zürich, 1996.
5. A. Henrich. A homogeneous access structure for standard attributes and document representations in vector space. In *Proc. 3rd Intl. Workshop on Next Generation Information Technologies and Systems*, Neve Ilan, Jerusalem, Israel, June 1997.
6. Portable Common Tool Environment - Abstract Specification / C Bindings / Ada Bindings. Standards ECMA-149/-158/-165, 3rd edition, and ISO IS 13719-1/-2/-3, 1994.
7. G. Salton. *Automatic Text Processing: the Transformation, Analysis, and Retrieval of Information by Computer.* Addison-Wesley, Reading, Mass., USA, 1989.
8. G. Salton and C. Buckley. Term-weighting approaches in automatic text retrieval. *Information Processing & Management*, 24(5):513–523, 1988.
9. G. Salton, A. Wong, and C.S. Yang. A vector space model for automatic indexing. *Communications of the ACM*, 18(11):613–620, November 1975.