Document Retrieval Facilities for Repository-Based System Development Environments

Andreas Henrich

Praktische Informatik, Fachbereich Elektrotechnik und Informatik
Universität Siegen, D-57068 Siegen, Germany
Email: henrich@informatik.uni-siegen.de

Abstract

Modern system development environments usually deploy the object management facilities of a so-called repository to store the documents created and maintained during system development. PCTE is the ISO and ECMA standard for a public tool interface for an open repository [23]. In this paper we present document retrieval extensions for an OQL-oriented query language for PCTE. The extensions proposed cover (1) pattern matching, (2) term based document retrieval with automatically generated document description vectors, (3) the flexible definition of what is addressed as a "document" in a given query, and (4) the integration of these facilities into a CASE tool. Whereas the integration of pattern matching facilities into query languages has been addressed by other authors before, the main contribution of our approach is the homogeneous integration of term based document retrieval and the flexible definition of documents.

1 Introduction

Repository-based applications are in wide-spread use in the domain of system development environments today. The intention is that all tools available for system development in a company should store the documents created in an open repository to enable data integration. As a side-effect a company wide information pool for system development is created. However, powerful query facilities are needed to exploit this information pool. Unfortunately PCTE – the ISO and ECMA standard for a public tool interface for an open repository [23, 30] – does not include such query facilities. Rather the usual access to objects is by navigation. PCTE has no facilities to state queries against the OMS in a descriptive way. To fill this gap, some proposals for query languages have been made [28, 1, 14], but these proposals are only concerned with traditional fact retrieval. Since the data stored in a repository is usually made up of documents like requirements documents, OOA-Diagrams, module specifications or source code, it seems to be natural to include document retrieval facilities into these query languages.

To overcome the mismatch between the query facilities and the data actually stored in the repository, we present document retrieval extensions for an OQL-oriented query language called P-OQL (PCTE-Object-Query-Language).

Although our considerations are based on this concrete environment, their applicability is by no means restricted to P-OQL or PCTE.

To become more precise, we have to clarify our understanding of the term *document*, which can be described by the definition given in [24]: *A document is a unified collection of information pertaining to a specific subject or related subjects.*

Obviously this is a relatively open understanding of a document. Examples for documents in this sense are: A paper like this one, an OOA-document representing the result of an Object-Oriented Analysis, the source code of a module, a bill, or a technical drawing.

The type of document retrieval we address in this paper is concerned with the textual information of the documents. This information can be addressed either using pattern matching facilities or using the term based methods proposed in the field of information retrieval.

For our considerations it is important that documents maintained in a repository, can be stored either at a coarse-grained level or at a fine-grained level.

At the *coarse-grained level*, a document directly relates to exactly one object. Beside other attributes like the author, the creation date or the last change date, such an object has one attribute of type *long field* containing the document itself. The document is usually stored in this attribute in a proprietary format, which can be interpreted only by the tool(s) especially designed to deal with this type of documents. E.g. in a software development environment, an OOA-document may be stored as a byte string which can be interpreted only by the OOA-editor. If another tool wants to access the OOA-document for some purpose, it has to use services provided by the OOA-editor. In this case, our retrieval system would require each tool managing documents to provide a method which yields an ASCII representation of the textual parts of the document which in turn could be used to apply our document retrieval facilities.

At the *fine-grained level*, the OOA-editor would use the data modeling facilities of the repository to structure the OOA-document, i.e. to store the document as a complex object consisting of several parts. For example each class, each method and each attribute could be stored as an object with corresponding attributes as depicted in the schema pre-

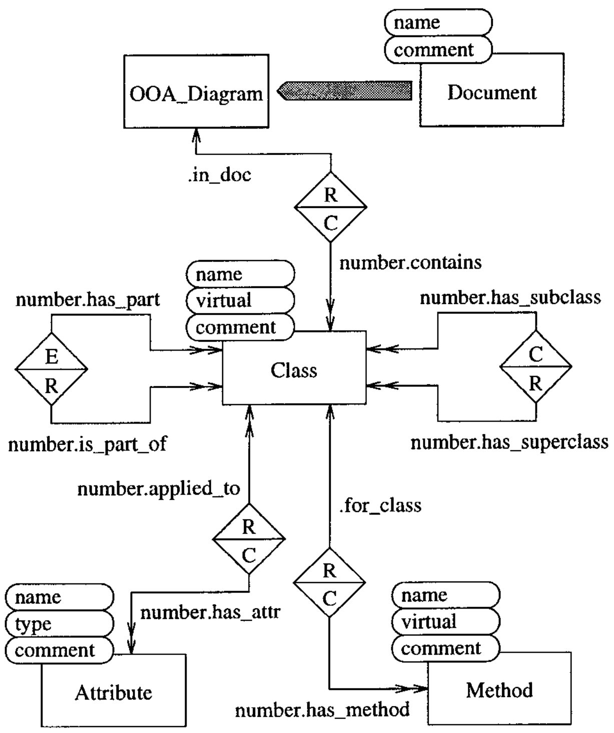

Figure 1: Example schema

sented in figure 1 which will be described in more detail in section 3. Here an OOA-document is modeled as a complex object which consists of class definitions. The class definitions themselves consist of the attributes *name*, *virtual* and *comment* and the corresponding attribute and method definitions which are stored as objects with respective attributes themselves. In this case the textual information of a document is stored as ASCII text in the string attributes of the objects forming the document.

Regardless of whether the data is modeled at a coarse-grained or at a fine-grained level, document retrieval should be possible. To this end, a retrieval system together with the associated query language has to deal with various topics:

1. *Addressing a document:* If coarse-grained modeling is used, addressing a document does not cause major problems because a document corresponds to exactly one object. On the other hand, using fine-grained modeling the situation is somewhat more complicated. In this case the query language should allow to address the whole textual information of a "document" which may be distributed over many objects of different types in a simple but nevertheless flexible way to apply document retrieval facilities. The techniques developed for this purpose will be explained in section 5.

2. *Performing document retrieval via pattern matching:* The query language should allow for queries using pattern matching facilities similar to the UNIX *grep*-command. To this end, we introduce an operator for P-OQL named *grep* which will be described in section 4.

3. *Performing term based document retrieval:* Besides full text retrieval using pattern matching, term based retrieval should be supported. Here each document is associated with a document description vector. Each component of this vector indicates the relevance of the document with respect to a given term. Then a query can be stated giving the terms of interest which may be weighted. In the field of information retrieval various techniques for the automatic creation of document description vectors and to calculate the similarity between a query vector and a document description vector have been proposed. In section 6 we will show how these techniques can be integrated into P-OQL.

4. *Integrating document retrieval and fact retrieval:* Obviously there should not be the alternative to do either document or fact retrieval, rather there should be an integration of both. In section 4 and section 6 we will show that our extensions allow for a flexible cooperation between fact and document retrieval.

5. *Building an easy and flexible end-user-interface:* As with all other powerful query languages writing a complex query in P-OQL is no simple task which can be performed by an unexperienced end user on the fly. Hence, we have built a front-end for our query language which allows to enter most document retrieval queries in an easy and straightforward manner and to key in complex queries directly in P-OQL. The concepts of this interface are sketched in section 7.

6. *Presenting the result:* In contrast to usual document retrieval we are not concerned with a homogeneous set of documents which can be presented to the user of the system using a standard document viewer. Rather we deal with a variety of documents and for each document type only the corresponding tool knows how to present the documents. Furthermore, a query may yield objects which do not directly have a corresponding editor, because they are not documents themselves, but parts of documents. In section 7 we will explain how these problems can be overcome when a retrieval component is integrated in a CASE environment.

Before dealing with the topics sketched above in more detail, section 2 will give a short survey of related work. Thereafter the concepts of PCTE and P-OQL which are essential for this paper are sketched in section 3.

2 Related Work

The work done in three related research areas has stimulated our work, namely (1) query languages for system development environments (2) path expressions to simplify navigation in object-oriented queries and (3) the integration of information retrieval concepts and databases.

An early approach towards query languages for system development environments is DQMCS presented in [28]. DQMCS is a reusable building block for tool writers that provides query facilities on the object base but incorporates no special document retrieval capabilities. In [18] first ideas for a query language are sketched together with more general considerations on an information retrieval common service for PCTE. In this work the need for a pattern matching facility is stressed as an important but open issue. Two interesting query languages for PCTE are presented in [1]. PNQL is an approach for a navigation-oriented query language using path expressions extended by predicates and output definitions. PQL has an SQL-like syntax, but is nevertheless largely navigation-oriented. Both languages do not contain document retrieval facilities.

As mentioned in the introduction the combination of fine-grained data modeling and document retrieval facilities brings up the need to define, which objects make up a document. One way to define such a set of objects is to use regular path expressions. Then all objects which can be reached from a given root object via paths matching a given path expression are considered as parts of the document. Many papers dealing with path expressions in object oriented databases have been published (see e.g. [12, 19, 20, 16, 3, 27, 10]). In contrast to our approach most of these papers use path expressions as some kind of abbreviation to specify a desired connection between two objects. Hence, some of them discuss several strategies which can be used when the connections are ambiguous. Instead, we use path expressions to define a set of objects. Furthermore, the data model of PCTE – where relations are normally bi-directional – enforces special considerations and prevents the direct application of existing approaches to path expressions in our environment.

Over the years a lot of work has been done integrating information retrieval concepts and database concepts. For example, Brown et al. [2] use a persistent object store to support the inverted file of a full-text information retrieval system. In earlier papers Croft et al. [6, 7] describe a loosely-coupled integration of a text retrieval system and an object-oriented database system. However, their focus is on the inference net model used as the retrieval model for the complex objects. They do not address the integration at the query language level nor do they discuss the flexible definition of the addressed "documents", rather they use a given part-of hierarchy. Fuhr [11] introduces a probabilistic algebra which is a generalization of the standard relational algebra. As with our approach this allows to combine document retrieval and standard database queries, but since the probabilistic algebra is based on relational algebra, it neglects many aspects of object-oriented data models.

In [22] and [13] approaches for the construction of software libraries based on information retrieval techniques are presented. Some of the ideas presented in these papers can be adapted to our system. For example in [22] an indexing scheme based on lexical affinities is used to identify a set of attributes for each document. However both papers do not deal with an integration at the query language level, or with fine-grained modeling of data.

In recent years also various systems introducing query languages for document retrieval have been proposed.

In [4] a mapping from SGML documents into OODBs and an extension of an SQL-like OODB query language in order to deal with SGML document retrieval is given. The query language extensions address pattern matching as well as querying paths. However, the integration of term based document retrieval and the flexible definition of documents needed in this context are not discussed. Furthermore the presented facilities for querying paths are not appropriate for the PCTE data model containing key attributes for links.

The same is true for the W3QS described in [21]. W3QS is a query system for the World-Wide Web and includes an SQL-like query language providing pattern matching facilities and graph patterns which can be used to address the structure of WWW documents. Since the "data model" of the WWW has nearly nothing in common with the PCTE data model, there are – and must be – a lot of syntactic and semantic differences compared to our query language.

In [5] HyperFile is presented as a data and query model for hypertext documents. HyperFile can be seen as an add-on system that enhances a DBMS for hypertext management. As with our approach links in the data can be used by queries to produce sets of objects in a navigating manner. The *arrow* and *setfilter* notations used in the HyperFile Interface Language (HIL) are somewhat similar to our regular path expressions. On the other hand, as with the other systems HIL does not include term based document retrieval and in contrast to our approach HyperFile is based on an extremely simple data model. Furthermore, HIL is a separate Prolog-like query language whereas our approach extends an OQL-oriented query language in a homogeneous way.

3 Example Environment

3.1 PCTE

As mentioned PCTE (Portable Common Tool Environment) is the ISO and ECMA standard for a public tool interface for an open repository [23, 30]. The PCTE object management system (OMS) can be envisaged as a generalization of the UNIX file system. Files are replaced by objects which may have contents of type long field. Arbitrary relationships between objects can be expressed by links. Attributes can be applied to objects and links. The usual access to objects and links is by navigation.

H-PCTE [17] — the PCTE implementation used as the basis for our implementation — is a main-memory-oriented high-performance implementation of the OMS of PCTE especially designed to meet the performance requirements arising from fine-grained data modeling.

The PCTE data model is structurally object-oriented and can be seen as an extension of the binary Entity-Relationship Model. The object base contains objects and relations. Relations are normally bi-directional. Each relation is realized by a pair of directed links, which are reverse links of each other, i.e. point into opposite directions.

- *Objects:* The type of an object is given by its name, a set of applied attribute types and a set of allowed outgoing link types. New object types are defined by inheritance, i.e. each new object type is a subtype of one or more existing types.

  Object references are the PCTE means to access objects. They are normally set by navigating operations.

- *Attributes:* An attribute type is given by an attribute name, a value type and an initial value. The permissible value types are boolean, string[1], integer, real, time and user defined enumeration types. Structured and set-valued attributes are not supported.

- *Links:* A link type is given by a name, an ordered set of attribute types called key attributes, a set of (non-key) attribute types, a set of allowed destination object types and a category. The category of a link type defines how instances of that link type will affect the behavior of certain PCTE primitives, e.g. if the deletion of a link leads to the deletion of its destination object. PCTE offers five 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

[1]According to the standard for PCTE, an object can also have *"a storage of data representing the traditional file concept"* called its *contents*. However, in H-PCTE *contents* are treated as normal string attributes, and the operations defined in the standard for contents are available in H-PCTE for string attributes.

and representing a property of the origin object), *implicit* (assuring referential integrity) and *designation* (without referential integrity).

Throughout this paper we will use the simplified schema for OOA-documents given in figure 1 in the typical PCTE schema diagram style as the basis for our examples. It consists of the object types *Document*, *OOA_Diagram*, *Class*, *Attribute* and *Method*. The attribute types applied to each object type are given in the ovals at the upper left corner of the rectangles representing the object types.

The link types representing the relationships between the object 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 attribute *number* is used for this purpose. A *'C'*, *'E'* or *'R'* in the triangles at the center of the line representing a pair of links, indicates that the link in the corresponding direction has category *composition*, *existence* or *reference*. In the example, each OOA-diagram contains one or more root classes as components, i.e. classes which do not have a superclass. Because the link types *contains*, *has_subclass*, *has_method* and *has_attr* have category *composition*, for an actual OOA-diagram all classes, methods and attributes contained in the graph originating from the object representing the OOA-diagram and spanned by links of these types, are regarded as components of the OOA-diagram.

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

3.2 P-OQL

P-OQL is an OQL-oriented query language for (H-)PCTE [14]. The main differences to standard OQL are due to the adaptation to the data model for PCTE. Hence, especially the treatment of links is specific to P-OQL.

Because of its similarity to OQL, P-OQL can be used very similar to SQL. For example, the following query would yield a bag containing the name of each class in the object base for which the *virtual* attribute is true:

```
select name from Class where virtual = true
```

A query in P-OQL is either a select-statement, or the application of an operator like *count*, *sort+*, *sort–*, or *union*. The following example computes the number of classes in the object base:

```
count (select name from Class)
```

In general, a select-statement consists of three main parts:

select <*target-clause*>
  from <*base-sets-clause*>
  where <*qualification-clause*>

In all three parts of a select-statement, it might be useful, to address objects or links relative to another object or link. To give an example, assume that we want to search for the name and the attributes of all classes in the OOA-diagram named *"Billing"* for which the instances can be used as parts:

```
select C:name,
       C:_.has_attr/->name
   from D in OOA_Diagram,
        C in (D:_.contains/[_.has_subclass]*/->.)
   where D:name = "Billing" and
         0 < count C:_.is_part_of->.
```

In the **base-sets-clause** of this query two base sets are defined: Base set *D* addressing all OOA-diagrams in the object base 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 `_.contains/[_.has_subclass]*`. The definition of base set *C* has the following interpretation:

- `D:` means that the actual element of base set *D* is used as starting point of the definition.
- `_.contains` means that exactly one link of type *contains* must be traversed. `'_'` is used as a wild-card for numerical key attributes denoting that arbitrary key values are allowed. In addition intervals can be specified for numerical key attributes and regular expressions can be used for string key attributes.
- `[_.has_subclass]*` means that zero or more links matching the link definition `_.has_subclass` have to be traversed. In fact, not only links can be used in the `[ ... ]*` construction but arbitrary path definitions (e.g. consisting of a sequence of link definitions separated by slashes). Furthermore P-OQL knows `[` *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.

  The semantics of these constructs is as follows: If an object is reached via the same *link definition* (which may be the last link definition in a `[` *path_definition* `]`, `[` *path_definition* `]+` or `[` *path_definition* `]*`) for the second time, the investigation of the path under concern is stopped without any further action. This assures termination and brings up the expected result.
- `/->` is used to address the destination object of the path. In addition `->` can be used to address the last link of the path.
- `.` means that the object (or link) under concern is addressed. It is also possible, to address an attribute – which is e.g. the case in the target-clause of the example query above – or to address a tuple containing several components (see [14] for more details).

It remains to mention that base set *C* in our example depends on base set *D*. Hence, for each element in *D* the expression `D:_.contains/[_.has_subclass]*/->.` defines a set of objects which are used as base set *C* for this element of base set *D*.

Summarizing, P-OQL knows five types of base sets:

1. *Object Types:* Since PCTE object types build an inheritance hierarchy, P-OQL allows to define the objects addressed by an object type in two ways:
   - *object_type_name* : only objects of this type
   - *object_type_name^* : all objects of this type and its descendant types
2. *Link Types:* Links of a given link type can be addressed by stating the link type name.
3. *Select-Statements:* The result of a select-statement can be used as base-set if it is either a set/bag of objects or a set/bag of links.

4. *Passed Sets of Objects or Links:* At the API of P-OQL the calling application can pass sets of objects and links as base sets which can be addressed in the query.

5. *Sets Defined via Regular Path Definitions:* As in our example, a regular path definition can be used as base-set if it addresses either objects or links.

In the **qualification-clause** of our example query two conditions are given. `D:name = "Billing"` means that the attribute *name* of the OOA-diagram under concern must have the value *"Billing"*. In the second condition the regular path expression `C:_.is_part_of->.` is used to address the set of links of type *is_part_of* originating from the class (element of base set C) under concern. The condition says, that the number of elements in this set must be greater than zero. Besides the simple predicates in the example query, P-OQL (like OQL) includes other features, like quantification, set operations or type tests.

The **target-clause** of the example query defines, that a bag of pairs is requested. Each pair consists of (1) the name of the class and (2) a bag with the names of the attributes of the class. In addition, arithmetical operations, set operations and sub-selects can be used in the target-clause.

4 Pattern Matching

As mentioned in the introduction, one way to perform document retrieval is to use pattern matching facilities. A typical example for this technique is the *grep* command in UNIX, which has proven to be extremely useful for everyday information retrieval problems as long as documents are stored as text files.

To allow similar operations on an object base, we integrated a new operator *grep* into P-OQL which can be used similar to the UNIX command. *grep* is realized as a binary operator, where the first operand must be a regular expression and the second operand must address a string attribute – other possible choices for the second operand will be discussed later on. The *grep* operator yields a set of quintuples. One quintuple for each line in the string attribute containing a pattern that matches the regular expression. Each quintuple has the following components: (1) an object reference or a link descriptor (reference to the origin object plus link name) for the object or link accommodating the attribute, (2) the name of the attribute, (3) the number of the line containing the pattern(s), (4) the contents of the line containing the pattern(s) and (5) a set with pairs containing the start position and the length for each pattern in the line that matches the regular expression.

Using this *grep* operator, the following query searches for the classes where the term *"Billing"* or *"billing"* occurs in the comment attribute:

```
select ('[Bb]illing' grep comment)
  from Class
  where 0 < count ('[Bb]illing' grep comment)
```

The result of this query is a bag of sets of quintuples. The qualification-clause assures that a set of quintuples is inserted into the result bag only for those classes for which the set of quintuples created by `'[Bb]illing' grep comment` is nonempty.

One remaining difference to the UNIX command is that the P-OQL query above considers only the *comment* attribute while the UNIX *grep* command addresses the whole textual information of a file. To overcome this remaining difference, we extended the *grep* operator such that it can be applied to objects and links as well. If it is applied to an object or a link, it is automatically applied to all string attributes of the object or link. Hence, the following query would search for patterns matching the regular expression `'[Bb]illing'` in all string attributes of the class objects – i.e. in the attributes *name* and *comment*:

```
select ('[Bb]illing' grep .)
  from Class
  where 0 < count ('[Bb]illing' grep .)
```

Since the *grep* operator is homogeneously integrated into P-OQL, it can be combined with all other features of P-OQL.

5 Addressing a document

Up to now we have assumed that we are searching for documents – class definitions – which are stored in exactly one object. However, if the data is modeled at a fine-grained level, this approach is not sufficient. If we take into consideration that a class definition in our example schema consists of the object of type *Class* and the methods and attributes connected to this class via *composition* links, it seems to be natural to regard all text attributes of the class itself and the referred attributes and methods as the textual description of the class.

This means that our document retrieval extensions must allow to address the text attributes of all objects building the document under concern as one operand for the operators dealing with the *"textual representation"* of a document. Hence, P-OQL had to be extended by new features which allow to deal with complex objects in an elegant way. These extensions have two aspects: (1) The operators used for document retrieval (such as the *grep* operator) have to deal with sets of objects representing one document. (2) Powerful features to define sets of objects which – in their entirety – should be considered as one document are needed.

The set of objects which build a class definition in the example can be addressed by the features of P-OQL presented in section 3.2 as follows:

```
([_.has_attr]/->. union _.has_method/->.)
```

In this expression `[_.has_attr]/->.` addresses the class itself as well as the associated attributes because passing a link of type *has_attr* is indicated to be optional using the `[ ... ]` construction. Hence, we get a set containing the class itself and the attribute definitions. Then we build the union of this set and the methods associated with the class via links of type *has_method*.

Now we extend the semantics of the *grep* operator to sets of objects: If a *grep* is applied to a set of objects, the result is the union of the results we achieve applying a *grep* to each element in this set. Using this extension, we can state the example query, searching for all class definitions for which the textual description – consisting of the class definition itself and the definitions of the corresponding attributes and methods – contains a pattern matching the regular expression `'[Bb]illing'`, as follows:

```
select ., ('[Bb]illing' grep ([_.has_attr]/->.
                              union
                              _.has_method/->.))
  from Class
  where 0 < count ('[Bb]illing' grep
                             ([_.has_attr]/->.
                              union
                              _.has_method/->.))
```

In this query a `'.'` is used as first component in the target-clause to get a pair consisting of an object reference for the class in the first component and the result of the *grep* operator in the second component. Each entry in the set produced by the *grep* operator exactly identifies the position of the matching pattern, because – according to the definition of the result of the *grep* operator in section 4 – it contains an object reference for the object which has the matching attribute and the attribute name.

The above query is a typical example for a situation, where it is useful to follow alternative path definitions. Since this seems to be a common situation, we introduced a new feature into P-OQL in order to allow for a simpler definition of such queries. We allow alternative path definitions to be given in a regular path expression using the following notation:

(*path_definition*$_1$ | ... | *path_definition*$_n$)

Although this brings up a somewhat easier notation, it does not introduce real new concepts. On the other hand the definition of cumbersome path definitions has been addressed in various research done in recent years. In a lot of articles like e.g. [12, 19, 20, 16, 3, 27] the authors state, that it would be much more convenient to specify simply that some type of connection must exist between two objects – i.e. that a path between these objects must exist. However as mentioned in [12] this strategy brings up severe problems if connections between objects are usually bi-directional – as is the case with the PCTE data model.

PCTE itself overcomes part of this problem using the link categories. For example there are operations in PCTE to pick up all links originating from an object. If the scope of this operation is set respectively, the operation returns all links originating from any component of the object under concern. Hence, the complex object built by composition links is regarded as the operand of the operation.

Our simple example which tries to address a class definition consisting of the class itself and the attached attribute and method definitions shows that simply applying this approach does not yield satisfactory results, because in our example schema not only the attribute definitions and the method definitions are components of a class definition, but also the subclass definitions together with their attribute and method definitions and their subclass definitions ...

Hence, we introduced a more powerful mechanism into P-OQL which includes addressing all components of an object as a special case.

In contrast to the link definitions used up to now, which have been based on a given link type name and a definition of the allowed values for the key attributes using wild-cards or intervals, we decided to allow the specification of a set of link categories instead, 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*. However, as mentioned above, this extension would not be sufficient to state the simple query given above. Therefore, a *shield* option has been introduced. This option allows to specify a set of object and/or link types which must not be traversed. E.g. `{c shield Class}` addresses all composition links except those for which the destination object is of type *Class*.

Using this feature, we can state our example query as follows:

```
select ('[Bb]illing' grep [{c shield Class}]/->.)
   from Class
   where 0 < count ('[Bb]illing'
                    grep
                    [{c shield Class}]/->.)
```

It remains to mention that not only the category of the link itself, but also the category of its reverse link can be specified. *'c'*, *'e'*, *'r'*, *'i'* and *'d'* can be used to specify that the link should be of category *composition*, *existence*, *reference*, *implicit* or *designation* and *'@c'*, *'@e'*, *'@r'* and *'@i'* can be used to specify that the reverse link of the link to be passed should be of category *composition*, *existence*, *reference* or *implicit*. This feature allows e.g. to ask for all documents containing at least one attribute where a pattern matching the regular expression `'[Aa]ccount'` occurs in the comment[2]:

```
select distinct D:.
   from A in Attribute, D in A:[{@c}]+/->.
   where 0 < count ('[Aa]ccount' grep A:comment)
         and D:. is of type Document^
```

In this query base set D addresses all objects which have the actual attribute – i.e. the actual element of base set A – as a component, and the qualification-clause restricts D to those objects which are of type *Document* or a respective subtype.

6 Term Based Document Retrieval

Besides pattern matching, a second way to process document retrieval queries is to compare description vectors for the stored documents and the query. To this end a vocabulary containing the terms which can be used to characterize the documents is created either automatically or manually. Techniques to create such a vocabulary automatically are e.g. described in [25].

If the vocabulary contains t terms, each document D can be represented by a document description vector $\mathcal{D} = (w_{d1}, w_{d2}, \ldots, w_{dt})$ where w_{dk} represents the relevance of document D with respect to term k. In the literature various term-weighting formulas are given to calculate the w_{dk} automatically. In the following, we use the formulas presented in [26]. Let N denote the number of documents, n_k denote the number of documents containing term k, and tf_{dk} denote the *term frequency* of term k in document D.

Then the components of the document description vector $\mathcal{D}$ can e.g. be calculated as follows:

$$w_{dk} = \frac{tf_{dk} \cdot \log \frac{N}{n_k}}{\sqrt{\sum_{i=1}^{t} \left(tf_{di} \cdot \log \frac{N}{n_i}\right)^2}} \qquad (1)$$

In this formula the term frequency is multiplied by the inverse collection frequency and normalized.

If the query itself is given as a text document Q for which similar documents are searched, the following formula proposed in [26] can be used to calculate the query description vector $\mathcal{Q} = (w_{q1}, w_{q2}, \ldots, w_{qt})$:

[2]In P-OQL a **select** normally creates a bag. If a set is desired, **select distinct** has to be used.

$$w_{qk} = \begin{cases} \left(0.5 + \frac{0.5\, tf_{qk}}{\max\limits_{1 \le i \le t} tf_{qi}}\right) \cdot \log \frac{N}{n_k} & \text{if} \quad tf_{qk} > 0 \\ 0 & \text{if} \quad tf_{qk} = 0 \end{cases} \tag{2}$$

Now the similarity between the query and a document in the object base can be calculated using for example the conventional vector product formula:

$$\text{similarity}(\mathcal{Q}, \mathcal{D}) = \sum_{k=1}^{t} w_{qk} \cdot w_{dk} \tag{3}$$

Using these formulas a typical query would be to search for a list of the 25 documents with the highest similarity values for a given query document Q sorted in descending order with respect to the similarity values.

To integrate such document retrieval facilities into P-OQL we introduced three new operators:

D_vector is a unary operator which calculates the document description vector $\mathcal{D}$ for a "document" D according to equation (1). The operand representing the document D can be either a string attribute, or a link (then the document is considered to be the concatenation of all string attributes of the link), or an object (then the document is considered to be the concatenation of all string attributes of the object), or a set of the above meaning that the "sub-documents" corresponding to the elements of the set are concatenated.

Q_vector is a unary operator which calculates the query description vector $\mathcal{Q}$ for a "query" Q according to equation (2). The operand is interpreted similar to the D_vector operator and may in addition be a string constant.

sim is a binary operator calculating the similarity of two vectors according to equation (3).

When applying formula (1) to calculate the result of a D_vector application, N and n_k $(1 \le k \le t)$ are calculated taking into concern all "documents" – defined according to the operand of the D_vector application – over the corresponding base set which fulfill the qualification-clause. If we take the query

```
select D_vector [{c shield Class}]*/->.
   from Class
   where virtual = false
```

as an example, this means that N is the number of non-virtual classes and n_k for $1 \le k \le t$ is the number of non-virtual classes for which the concatenation of the string attributes of the class definition itself and the string attributes of the associated attribute and method definitions contains the term k at least once. The values for N and n_k $(1 \le k \le t)$ when applying formula (2) to calculate the result of a Q_vector application are chosen with respect to the context. If the Q_vector application occurs as one operand of a sim operator and the other operand is a D_vector application – which is the usual situation – the respective values of the D_vector application are used. Otherwise, default values calculated when generating the vocabulary are used[3].

[3]The results presented in [29] suggest that it is not necessary to use the exact values for N and n_k in dynamic information retrieval systems. Therefore techniques to compute approximations for N and n_k seem to be an interesting research field.

Using these operators, we can e.g. search for documents similar to a given OOA-diagram named *"Accounting"*:

```
head[25]
  sort- (
    select (Q_vector Q:[{c}]*/->.
            sim
            D_vector D:[{c}]*/->.),
            D:.
       from Q in OOA_diagram, D in document^
       where Q:name = "Accounting" and
             Q:. != D:. )
```

In this query base set Q is used to address the query document. To this end, base set Q is restricted to the OOA-diagram named *"Accounting"* in the qualification-clause. The select-statement yields a bag of pairs, where the first component is the similarity value for the actual document in base set D and the second component is an object reference for this document. Since we use document^ as the base set for the documents, we are not only concerned with OOA-diagrams, but with all types of documents stored in the object base. The sort- operator is applied to the result of the select-statement yielding a list with the pairs sorted according to their similarity values. Then the head operator is used to extract the 25 elements at the beginning of this list. The condition Q:. != D:. in the qualification-clause of the select-statement is used to extract the query document from the result.

Taking a query document from the object base is only one way to proceed. Another opportunity would be to state the query text directly in the query:

```
head[25]
  sort- (
    select (Q_vector "This is the query text on
                      OMS and OQL"
            sim
            D_vector [{c}]*/->.),
            .
       from document^ )
```

Although stating a query in this way is possible, it may not really be a good choice, because equation (2), which is applied using the Q_vector operator, has originally been designed for real text documents but not for a short query formulation. This becomes obvious from the fact, that the term frequency in the query text is used to calculate the weight of the terms. Therefore it might be a better choice to state the query description vector itself. Let us assume, that term 3 in our vocabulary is *"OMS"*, term 7 is *"OQL"* and term 15 is *"PCTE"* [4]. Then the following query would search for documents concerned with these topics and state, that the relevance for *"PCTE"* is only half as important as the relevance for the other topics:

```
head[25]
  sort- (
    select ((0.0 | (3: 1.0), (7: 1.0), (15: 0.5))
            sim
            D_vector [{c}]*/->.),
            .
       from document^ )
```

[4]The terms in the vocabulary as well as the position of a given term can be requested from the *vocabulary manager*.

Note that the query description vector is given in the sparse vector representation supported by P-OQL. Here the 0.0 given before the '|'-sign is the default value used for all components except those which are explicitly defined in the remainder of the vector definition.

It has to be mentioned that the described operators can also be used to find e.g. pairs of similar documents. As an example the following query retrieves the 25 pairs of classes with highest similarity:

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

In this query two base sets with classes are used to address all pairs of classes. The pairs for which both components represent the same class are excluded in the qualification-clause. In the target-clause the similarity of the classes is calculated and an object reference for each class is requested. We have used a similar query to detect redundant definitions in the data element catalog of a large bank and achieved impressive results.

One main advantage of the integration of document retrieval functionality into a flexible query language is that arbitrary combinations of fact retrieval, and different types of document retrieval can be used in one query. An example is the following query searching for documents dealing with *"OMS"*, *"OQL"* and *"PCTE"* which do not contain the pattern *"C++"* and contain at least one object of type *Attribute*:

```
head[25]
  sort- (
    select ((0.0 | (3: 1.0), (7: 1.0), (15: 0.5))
            sim
            D_vector [{c}]*/->.),
           .
       from document^
       where 0 = count ('C++' grep [{c}]*/->.)
             and exists O
                   in [{c}]*/->.
                   with O:. is of type Attribute)
```

7 Integrating the Facilities into a CASE Tool

In the previous sections two points should have become obvious. (1) Together with the document retrieval extensions presented in this paper P-OQL is a powerful aid for the retrieval in a repository. (2) Unfortunately writing a complex query in P-OQL is no trivial task at all.

Hence, when integrating P-OQL into the Software Development Environment *Toolframe* [8], we had to design a user interface which allows, to state frequently used queries in a straightforward manner and at the same time to exploit all features of P-OQL when necessary[5]. For this purpose we integrated a retrieval tool called *Finder* into *Toolframe* in the following way.

[5]Besides this interactive user interface P-OQL can of course be used via an API. This allows tool developers to realize e.g. integrity checks or tool specific query functionality in an easy declarative way. See [14] for more details.

At the root window of Toolframe the available tools can be invoked. One possible choice is to call the *Finder*. Then the main window of the Finder is presented and the user can choose whether he wants to (1) key in a P-OQL query directly or (2) use a predefined form to define a pattern matching query or (3) use a menu based system to state a term based retrieval or (4) load a query, which has been created before using one of the other choices and stored for future use. In the pattern matching case for example, the user has to key in a regular expression, he has to select the object types which have to be considered, and he can define whether the components of an object should also be taken into account or not.

Irrespective of which input facility has been used, the system generates a P-OQL query which is processed by the P-OQL query evaluator. Then the result of the query is presented by a standard result viewer. Here all values except of link 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 Finder first of all creates a list of all objects containing the object under concern as a component. In this list the objects are sorted with respect to the length of the paths of composition links connecting the object in the list and the object under concern. Then Toolframe 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 tools which can be applied to the object itself – if any – are presented at the top of the selection list, and the user can choose the tool he wants to start 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 Toolframe for the tools available for the actual object and the user can start these tools from the menu bar. Since queries can be saved, experienced users can compose queries for different purposes, which can be used by other users.

8 Conclusion

We have presented document retrieval extensions for an OQL-oriented query language for the ISO and ECMA standard PCTE in this paper. The main contributions made are the homogeneous integration of term based retrieval facilities into the query language and powerful features for the flexible definition of the "documents" addressed by the retrieval facilities in a given query. Furthermore, we think that the realization of document retrieval facilities for the PCTE standard is interesting in its own right because of the semantically rich data model of PCTE and because of the importance of PCTE as an international standard for repositories.

Whereas the integration at the query language level has proven to be practically applicable in our experiments, a lot of open topics remain:

First of all, optimization and index structures are an important research direction. We try to use a multi-attribute index structure [15] for standard attributes and document description vectors at the same time. Alternatively we try to use conventional B-trees for the standard attributes and inverted lists for the document description vectors exploiting the results presented in [9] and [2]. To perform effective query optimization in such an environment, a sophisticated cost model is needed.

With respect to term based retrieval a crucial aspect is the vocabulary. At the moment we use one vocabulary for the whole object base, which brings up a vocabulary with a large number of terms which in turn causes problems especially with the index structures. Therefore we will try to use project specific vocabularies in the future. This will help keeping the vocabularies smaller, but will cause problems with *inter-project queries*.

Since we do not deal with large text documents but with short descriptions and comments in the typical application area of our system – i.e. in system development environments – it will be important to check which methods and formulas for term based document retrieval proposed in the literature are best suited for this particular application area. The results presented in [31] might e.g. be useful in this respect.

References

[1] B. Bird. An Open Systems SEE Query Language. In *Proc. 7th Conf. on Software Engineering Environments*, pages 34–47, Noordwijkerhout, Netherlands, 1995.

[2] E.W. Brown, J.P. Callan, W.B. Croft, and J.E.B. Moss. Supporting Full-Text Information Retrieval with a Persistant Object Store. In *Proc. 4th Intl. Conf. on Extending Database Technology*, volume 779 of *LNiCS*, pages 365–378, Cambridge, UK, 1994.

[3] J. Van den Bussche and G. Vossen. An Extension of Path Expressions to Simplify Navigation in Object-Oriented Queries. In *Deductive and Object-Oriented Databases*, volume 760 of *LNiCS*, pages 267–282, Phoenix, Ariz., USA, 1993.

[4] V. Christophides, S. Abiteboul, S. Cluet, and M. Scholl. From Structured Documents to Novel Query Facilities. In *Proc. ACM SIGMOD Intl. Conf. on Management of Data*, pages 313–324, Minneapolis, Minn., USA, 1994.

[5] C. Clifton, H. Garcia-Molina, and D. Bloom. Hyperfile: A data and query model for documents. *VLDB Journal*, 4(1):45–86, 1995.

[6] W.B. Croft, L.A. Smith, and H.R. Turtle. A Loosely-Coupled Integration of a Text Retrieval System and an Object-Oriented Database System. In *Proc. 15th Annual Intl. ACM SIGIR Conf. on Research and Development in Information Retrieval*, pages 223–232, Copenhagen, Denmark, 1992.

[7] W.B. Croft and H.R. Turtle. Retrieval of Complex Objects. In *Proc. 3rd Intl. Conf. on Extending Database Technology*, volume 580 of *LNiCS*, pages 217–229, Vienna, Austria, 1992.

[8] D. Däberitz and U. Kelter. Rapid prototyping of graphical editors in an open SDE. In *Proc. 7th Conf. on Software Engineering Environments*, pages 61–72, Noordwijkerhout, Netherlands, 1995.

[9] C. Faloutsos and H.V. Jagadish. Hybrid Index Organizations for Text Databases. In *Proc. 3rd Intl. Conf. on Extending Database Technology*, volume 580 of *LNiCS*, pages 310–327, Vienna, Austria, 1992.

[10] J. Frohn, G. Lausen, and H. Uphoff. Access to Objects by Path Expressions and Rules. In *Proc. 20th Intl. Conf. on Very Large Data Bases*, Santiago de Chile, 1994.

[11] N. Fuhr. A Probabilistic Relational Model for the Integration of IR and Databases. In *Proc. 16th Annual Intl. ACM SIGIR Conf. on Research and Development in Information Retrieval*, pages 309–317, Pittsburgh, Penn., USA, 1993.

[12] C. Harrison. An Adaptive Query Language for Object-Oriented Databases: Automatic Navigation Through Partially Specified Data Structures. Technical Report NU-CCS-94-19, Northeastern University College of Computer Science, October 1994.

[13] S. Henninger. Retrieving Software Objects in an Example-Based Programming Environment. In *Proc. 14th Annual Intl. ACM SIGIR Conf. on Research and Development in Information Retrieval*, pages 251–260, Chicago, Ill., USA, 1991.

[14] A. Henrich. P-OQL: an OQL-Oriented Query Language for PCTE. In *Proc. 7th Conf. on Software Engineering Environments*, pages 48–60, Noordwijkerhout, Netherlands, 1995.

[15] A. Henrich and J. Möller. Extending a spatial access structure to support additional standard attributes. In *Proc 4th Intl. Symposium on Advances in Spatial Databases*, volume 951 of *LNiCS*, pages 132–151, 1995.

[16] Y.E. Ioannidis and Y. Lashkari. Incomplete Path Expressions and their Disambiguation. In *Proc. ACM SIGMOD Intl. Conf. on Management of Data*, pages 138–149, Minneapolis, Minn., USA, 1994.

[17] Udo Kelter. H-PCTE - A High-Performance Object Management System for System Development Environments. In *Proc. 16th Annual Intl. Computer Software and Applications Conf.*, Chicago, Ill., USA, 1992.

[18] Udo Kelter. An Information Retrieval Common Service Based on H-PCTE. In *Proc. 6th Conf. on Software Engineering Environments*, pages 101–108, Reading, UK, 1993.

[19] M. Kifer, W. Kim, and Y. Sagiv. Querying Object-Oriented Databases. In *Proc. ACM SIGMOD Annual Conf. on Management of Data*, pages 393–402, San Diego, Cal., USA, 1992.

[20] W. Kim. Observations on the ODMG-93 Proposal for an Object-Oriented Database Language. *SIGMOD Record*, 23(1):4–9, 1994.

[21] D. Konopnicki and O. Shmueli. W3QS: A Query System for the World-Wide Web. In *Proc. 21th Intl. Conf. on Very Large Data Bases*, pages 54–65, Zürich, Switzerland, 1995.

[22] Y.S. Maarek, D.M. Berry, and G.E. Kaiser. An Information Retrieval Approach For Automatically Constructing Software Libraries. *IEEE, Transactions on Software Engineering*, 17(8):800–813, 1991.

[23] 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.

[24] J.M. Rosenberg. *Dictionary of computers, information processing, and telecommunications.* John Wiley & Sons, 2nd edition, 1987.

[25] G. Salton. *Automatic Text Processing: the Transformation, Analysis, and Retrieval of Information by Computer.* Addison-Wesley, Reading, Mass., USA, 1989.

[26] G. Salton and C. Buckley. Term-Weighting Approaches in Automatic Text Retrieval. *Information Processing & Management*, 24(5):513–523, 1988.

[27] E. Sciore. Query Abbreviation in the Entity-Relationship Data Model. *Information Systems*, 19(6):491–511, 1994.

[28] M. Tedjini, I. Thomas, G. Benoliel, F. Gallo, and R. Minot. A query service for a software engineering database system. In *Proc. 4th ACM Symp. on Software Development Environments*, ACM SIGSOFT Newsletter 15:6, pages 238–248, December 1990.

[29] C.L. Viles and J.C. French. On the Update of Term Weights in Dynamic Information Retrieval Systems. In *Proc. 4th Intl. Conf on Information and Knowledge Management*, pages 167–174, Baltimore, Md., USA, 1995.

[30] L. Wakeman and J. Jowett. *PCTE - The standard for open repositories.* Prentice Hall, Hemel Hempstead, Hertfordshire, UK, 1993.

[31] R. Wilkinson. Effective Retrieval of Structured Documents. In *Proc. 17th Annual Intl. ACM SIGIR Conf. on Research and Development in Information Retrieval*, pages 311–317, Dublin, Ireland, July 1994.