Zweitveröffentlichung

Henrich, Andreas

Retrieval-Dienste für Software-Entwicklungsumgebungen

Datum der Zweitveröffentlichung: 05.03.2025

Akzeptiertes Manuskript (Postprint), Konferenzveröffentlichung

Persistenter Identifikator: urn:nbn:de:bvb:473-irb-1069015

Erstveröffentlichung

Henrich, Andreas (1997): Retrieval-Dienste für Software-Entwicklungsumgebungen, in: Andreas Oberweis und Harry M. Sneed (Hrsg.), Software-Management '97 : Fachtagung der Gesellschaft für Informatik e. V. (GI), Oktober 1997 in München, Stuttgart ; Leipzig: Teubner, S. 147–162, doi: 10.1007/978-3-322-85166-6_10.

Retrieval-Dienste für Software-Entwicklungsumgebungen

Andreas Henrich

Abstract

Moderne Software-Entwicklungsumgebungen bestehen aus einer Vielzahl von Werkzeugen, die hinsichtlich der Benutzungsoberfläche, der Ablaufsteuerung und der Datenhaltung integriert sind. Die integrierte Datenhaltung wird dabei i. allg. durch die Verwendung eines Repository unterstützt. Sowohl die kommerziell verfügbaren Repositories als auch die Standards zu diesem Bereich stellen aber leider nur unzureichende Anfragemöglichkeiten zu den verwalteten Daten zur Verfügung. Deshalb haben wir beispielhaft Retrieval-Dienste für den ISO-Standard PCTE entwickelt, die Möglichkeiten des navigierenden Zugriffs mit der Funktionalität einer OQL-artigen Anfragesprache und Techniken des Dokumenten-Retrieval kombinieren. Das vorliegende Papier beschreibt diese Retrieval-Dienste und stellt exemplarisch einige Anwendungen vor.

1. Einleitung

Die Entwicklung komplexer Software-Systeme ist heute ohne entsprechende rechnergestützte Werkzeuge nicht mehr denkbar. Beispiele für derartige Werkzeuge sind Editoren oder Werkzeuge zur Überprüfung von Konsistenzbedingungen. Die Entwicklung geht dabei, wie Nagl [Nag93] es formuliert, *„seit den 80er Jahren von einzelnen Werkzeugen zu größeren Werkzeugansammlungen, von isolierten Werkzeugen zu integrierten Umgebungen, von unspezifischen zu semantischen Werkzeugen"*. Man spricht in diesem Zusammenhang von Software-Entwicklungsumgebungen, die nicht nur eine Ansammlung verschiedener Werkzeuge, sondern eine integrierte Arbeitsumgebung sein wollen.

Um die Integration der Werkzeuge zu unterstützen, wurden *Integrationsrahmen* für Software-Entwicklungsumgebungen definiert. Ein wichtiges Beispiel hierfür ist der ISO- und ECMA-Standard PCTE (Portable Common Tool Environment) [PCT94]. Ein wesentlicher Bestandteil des Integrationsrahmens ist die Datenverwaltungs-Komponente, die auch als *Repository* bezeichnet wird. Im Falle von PCTE handelt es sich dabei um ein operational objektorientiertes

Objekt-Management-System (OMS). Der Standard definiert für dieses OMS zahlreiche Aspekte — wie die DDL, den navigierenden Zugriff auf die Daten, Schema-Operationen, Versionierung, Replikation und Verteilung. Ein deskriptiver, mengenorientierter Zugriff auf die Daten ist aber ebensowenig vorgesehen wie eine inhaltsbasierte Dokumentensuche.

Andererseits gibt es zahlreiche Anwendungen, bei denen ein solcher mengenorientierter Zugriff eine kompakte Definition der benötigten Daten erlauben würde. Beispiele hierfür sind die Suche nach inkonsistenten Objekten, die Aufbereitung von Reports oder die Anwendung von Metriken. Wir haben daher speziell auf die Erfordernisse des dargestellten Anwendungsbereichs zugeschnittene Retrieval-Dienste entwickelt, die den im PCTE-Standard definierten navigierenden Zugriff auf die Daten mit der Option OQL-artiger Anfragen und mit Techniken aus dem Bereich des Dokumenten-Retrieval kombinieren.

Ausgangspunkt unserer Überlegungen ist H-PCTE [Kel92], eine hochperformante, hauptspeicherorientierte Implementierung des PCTE-OMS. Auf dieser Basis haben wir eine an das Datenmodell von PCTE angepaßte OQL-artige Anfragesprache definiert, die Techniken der Dokumentensuche aus dem Information-Retrieval integriert. Diese P-OQL genannte Anfragesprache werden wir in Abschnitt 2. beschreiben. Einen weiteren Schwerpunkt unserer Arbeiten, den wir in Abschnitt 2.6. kurz darstellen werden, bildet die effiziente Implementierung dieser Anfragesprache auf Basis entsprechender Zugriffsstrukturen. Schließlich werden wir in Abschnitt 3. exemplarisch einige Werkzeuge vorstellen, die mit Hilfe der beschriebenen Anfragesprache realisiert worden sind. Abbildung 1 faßt den Aufbau einer Software-Entwicklungsumgebung (SEU) im betrachteten Szenario zusammen und verdeutlicht die Einordnung der in den folgenden Abschnitten beschriebenen Teilaspekte.

An dieser Stelle soll noch kurz auf einige vergleichbare Ansätze hingewiesen werden. Ein erster Vorschlag im Hinblick auf eine Anfragesprache für System-Entwicklungsumgebungen ist DQMCS [TTB+90]. DQMCS kann als wiederverwendbarer Baustein für Werkzeugentwickler verstanden werden, der zur Implementierung von Anfragediensten verwendet werden kann. In [Kel93a] werden einige Anforderungen an Retrieval-Dienste skizziert. Dabei wird insbesondere die Notwendigkeit von Dokumenten-Retrieval-Funktionalitäten hervorgehoben. Zwei einfache Anfragesprachen für PCTE werden in [Bir95] beschrieben. Während eine dieser Sprachen Navigationsbefehle um Ausgabedefinitionen ergänzt, hat die andere eine SQL-artige Syntax. Sie orientiert sich

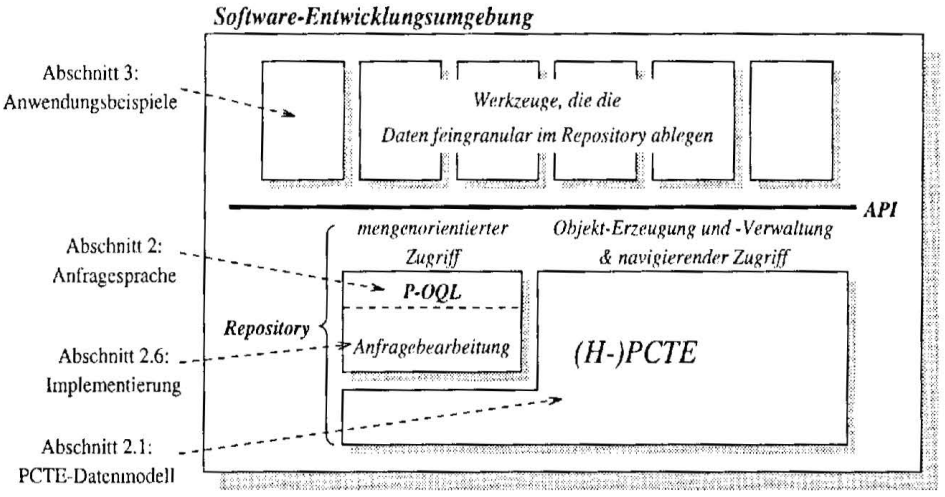

Abbildung 1: Aufbau einer SEU im betrachteten Szenario

aber trotzdem stark an navigierenden Operationen.

Während die obigen, auf System-Entwicklungsumgebungen ausgerichteten Ansätze den Dokumenten-Retrieval-Aspekt vernachlässigen, gibt es zahlreiche Ansätze zur Kombination mengenorientierter Anfragesprachen mit Dokumenten-Retrieval-Funktionalitäten im Umfeld von SGML, WWW oder Hypertextsystemen (vgl. hierzu z.B. [KS95, CACS94, CGMB95, VAB96]). Diese Ansätze gehen aber i. allg. von schwach strukturierten Daten aus, während wir auf dem ausdrucksstarken Datenmodell von PCTE aufsetzen. Daneben gibt es auch im Bereich des Information-Retrieval einige Ansätze zur Integration von Datenbanken und Dokumenten-Retrieval (vgl. z.B. [Fuh93]) und zur Unterstützung von Software-Bibliotheken (vgl. z.B. [MBK91] oder [Hen91]), die hier aber aus Platzgründen nicht im Detail betrachtet werden können.

2. Die Anfragesprache

Als Ausgangsbasis für die P-OQL[1] (PCTE Object Query Language) genannte Anfragesprache wurde OQL [Cat93] gewählt. Für den beabsichtigten Anwendungsbereich mußte OQL aber zum einen an das Datenmodell von PCTE angepaßt und zum anderen um Möglichkeiten des Dokumenten-Retrieval ergänzt werden. Im folgenden beschreiben wir daher kurz das Datenmodell von PCTE bevor wir auf die Besonderheiten von P-OQL eingehen.

[1] Eine ausführliche Darstellung einer ersten Version von P-OQL findet sich in [Hen95].

2.1. Das Datenmodell von PCTE

Das Datenmodell von PCTE kann als ein erweitertes Entity-Relationship-Modell beschrieben werden. Die Objektbank enthält Objekte und Beziehungen. Die Beziehungen sind i.d.R. bidirektional, d.h. jede Beziehung wird durch ein Paar aus zwei gegenläufigen, gerichteten *Links* repräsentiert.

Ein *Objekttyp* ist dabei gegeben durch eine Menge von Attributen, eine Menge von zulässigen ausgehenden Linktypen und eine Menge von direkten Elterntypen. Ein *Attribut* ist definiert durch seinen Namen (Attributname) und seinen Wertebereich (Attributtyp; z.B. *natural* oder *string*). Ein *Linktyp* ist definiert durch seinen Namen, eine geordnete Menge von Schlüsselattributen, eine Menge von Nicht-Schlüsselattributen, eine Menge der erlaubten Ziel-Objekttypen und eine Kategorie, die gewisse semantische Eigenschaften des Links festlegt. Beispielsweise wird für Links der Kategorie *reference* die referentielle Integrität (also die Existenz des Zielobjekts) garantiert und bei Links der Kategorie *composition* gilt das Zielobjekt als Komponente des Ausgangsobjekts.

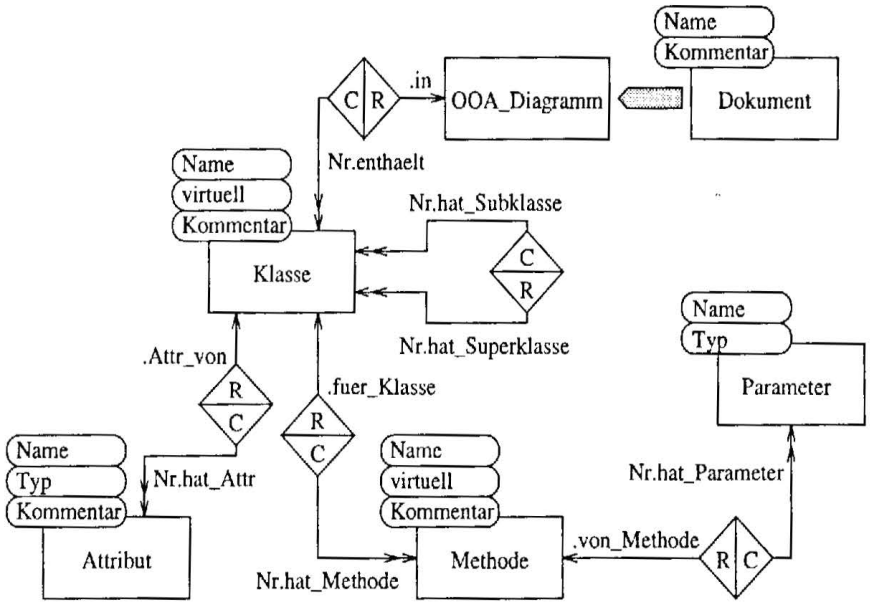

Abbildung 2: Beispiel-Schema für OOA-Diagramme

Abbildung 2 beschreibt ein einfaches PCTE Schema für OOA-Diagramme. Objekttypen sind durch Rechtecke und die zugehörigen Attribute durch Ovale repräsentiert. Die Linktypen, die die Beziehungen zwischen den Objekten repräsentieren, sind durch Pfeile dargestellt. Eine doppelte Pfeilspitze zeigt an,

daß der Link die Kardinalität *viele* hat. Links der Kardinalität *viele* müssen in PCTE über ein Schlüsselattribut verfügen. In unserem Beispiel wird für diesen Zweck immer das numerische Attribut *Nr* verwendet. Ein *C* oder *R* in den Dreiecken in der Mitte der Linie, die ein Paar von Links darstellt, zeigt dabei an, daß der Link in der entsprechenden Richtung die Kategorie *composition* oder *reference* hat. Schließlich enthält das Schema eine Vererbungsbeziehung zwischen den Objekttypen *Dokument* und *OOA_Diagramm*.

2.2. Die Darstellung des Ergebnisses

Eine wichtige Anforderung an Anfragesprachen ist die Forderung nach Abgeschlossenheit über dem jeweiligen Datenmodell. Da PCTE aber keine Mengen oder Listen kennt, haben wir zur Darstellung des Ergebnisses einer P-OQL-Anfrage einen abstrakten Datentyp (ADT) Value eingeführt, der einen PCTE Attributwert, eine Objektreferenz[2], einen Linkdeskriptor oder eine Menge, Multimenge, Liste oder Struktur von Values aufnehmen kann. Durch die Integration von Objektreferenzen und Linkdeskriptoren in diesen ADT bleiben wesentliche Vorteile einer abgeschlossenen Anfragesprache erhalten.

2.3. Berücksichtigung der Links

Im Verhältnis zum Datenmodell der ODMG, auf dem OQL basiert, muß eine an PCTE angepaßte Anfragesprache die ausdrucksstärkeren Beziehungen (Links) reflektieren. In P-OQL können deshalb nicht nur Objekttypen, sondern auch Linktypen zur Definition der Basismengen einer Anfrage verwendet werden.

So bestimmt die Anfrage '`select ^Name, /Name from hat_Subklasse`' alle Vererbungsbeziehungen zwischen Klassen. Mit dieser Anfrage wird zu jedem Link des Typs *hat_Subklasse* der Name der Superklasse und der Name der Subklasse bestimmt (durch ein vorangestelltes '^' bzw. '/' bezieht man sich auf das Ausgangs- bzw. Zielobjekt eines Links).

Eine weitere Berücksichtigung der Links erfolgt durch die Integration mächtiger regulärer Pfadausdrücke[3] in die Sprache. Damit können von einem Ob-

[2]Objektreferenzen und Linkdeskriptoren werden in PCTE zum navigierenden Zugriff auf Objekte und Links verwendet und mit Hilfe navigierender Operationen gesetzt.

[3]P-OQL orientiert sich hier zum Teil an den in der Literatur präsentierten Ansätzen zu Pfadausdrücken (vgl. z.B. [BV93, Sci94, FLU94]). Während in diesen Papieren Pfadausdrücke aber oftmals als eine Art Abkürzung verwendet werden, um eine gesuchte Beziehung zu definieren, definieren Pfadausdrücke in P-OQL eine Multimenge von Values.

jekt oder Link aus entfernte Informationen angesprochen werden. Ein Beispiel hierfür ergibt sich, wenn überprüft werden soll, ob die in der Objektbank abgelegten Vererbungsbeziehungen zyklenfrei sind. Die folgende Anfrage sucht nach den Namen aller Klassen, die in einem Zyklus enthalten sind:

```
select Name from Klasse where . in [_.hat_Subklasse]+/->.
```

Durch den Punkt in der `where`-Klausel wird die aktuelle Klasse (d.h. das entsprechende Objekt) adressiert. Der Pfadausdruck '`[_.hat_Subklasse]+/->.`' bestimmt die Menge der direkten und indirekten Subklassen der aktuellen Klasse. '`_.hat_Subklasse`' bedeutet dabei, daß Links des Typs *hat_Subklasse* mit beliebigen Schlüsselattributwerten verfolgt werden sollen. Durch den Einschluß dieser Linkdefinition in das Iterationskonstrukt '`[...]+`' wird definiert, daß Pfade verfolgt werden sollen, die aus einem oder mehreren Links bestehen, die der Linkdefinition entsprechen. Dem Iterationskonstrukt liegt dabei eine entsprechende Fixpunktsemantik zugrunde.

2.4. Integration mit dem navigierenden Zugriff

Um Objektreferenzen und Linkdeskriptoren, die durch navigierende PCTE-Operationen bestimmt worden sind, als Ausgangspunkt einer P-OQL-Anfrage zu verwenden, können *Sequenzen* (ein von PCTE zur Verwaltung von Mengen definierter ADT) mit Objektreferenzen oder Linkdeskriptoren über die Programmierschnittstelle von P-OQL übergeben und in Anfragen als Basismengen verwendet werden. Ferner wird immer dann, wenn in der `select`-Klausel einer Anfrage durch einen Punkt ein Objekt (bzw. Link) adressiert wird, eine entsprechende Objektreferenz (bzw. ein entsprechender Linkdeskriptor) in das Ergebnis der Anfrage aufgenommen, die als Ausgangspunkt für navigierende PCTE-Operationen dienen kann. So liefert die Anfrage '`select Name, . from Klasse`' zu jeder Klasse den Namen und eine Objektreferenz.

2.5. Möglichkeiten des Dokumenten-Retrieval

Die in einem Repository verwalteten Objekte haben oft den Charakter von Dokumenten mit einem relativ hohen Textanteil – z.B. in Form von als Freitext abgelegten Kommentaren. Zur Suche nach solchen Dokumenten bieten sich Techniken des Dokumenten-Retrieval an. Dabei ist nicht nur an die reine Zeichenkettensuche (analog zum UNIX-Befehl `grep`) zu denken, sondern z.B. auch an die Suche mit Hilfe des Vektorraummodells [SWY75]. In diesem

Modell werden Dokumente und Informationswünsche durch Beschreibungsvektoren dargestellt, auf denen ein Ähnlichkeitsmaß definiert ist. Den Ausgangspunkt bildet ein Vokabular, das die für potentielle Informationswünsche relevanten Begriffe enthält [4]. Zu den einzelnen Dokumenten wird dann die Relevanz bezüglich der Begriffe aus dem Vokabular bestimmt. Diese Relevanzwerte werden in einen Beschreibungsvektor für das Dokument eingetragen, der für jeden Begriff im Vokabular eine Komponente enthält. Man kann dann zu einem durch einen Anfragetext definierten Informationswunsch ebenfalls einen Beschreibungsvektor erzeugen, und Dokumente mit einem ähnlichen Beschreibungsvektor im Datenbestand suchen.

Wir haben in P-OQL beide Arten der Suche nach Dokumenten integriert. Für die zeichenkettenbasierte Suche enthält P-OQL einen an das UNIX Kommando *grep* angelehnten binären Operator, der als ersten Operanden einen regulären Ausdruck und als zweiten Operanden eine Dokumentdefinition erwartet. Dieser Operator erzeugt eine Menge, die für jede Zeichenkette innerhalb des Dokumentes, die dem regulären Ausdruck entspricht, einen Eintrag enthält. Als Dokumentdefinition können dabei nicht nur String-Attribute, sondern auch Objekte, Links sowie Mengen, Multimengen, Listen oder Strukturen verwendet werden. Der *grep* Operator bezieht sich dabei immer auf die textuellen Anteile der Kollektionen bzw. auf die String-Attribute der Objekte und Links.

Zur Ähnlichkeitssuche nach dem Vektorraummodell enthält P-OQL einen Operator `D_vector`, der zu einem analog zum *grep* Operator definierten Dokument einen Beschreibungsvektor erzeugt. Die Ähnlichkeit zwischen zwei Beschreibungsvektoren kann dann mit Hilfe des `sim`-Operators bestimmt werden.

Gesucht seien z.B. Klassendefinitionen, die zu einem textuell beschriebenen Informationswunsch relevant sind:

```
head[25]
  sort-
    (select (D_vector "<als Text formulierter Informationswunsch>"
            sim
            D_vector (., _.hat_Methode/->., _.hat_Attr/->.),

        from Klasse)
```

Die durch '`(., _.hat_Methode/->., _.hat_Attr/->.)`' definierten Dokumente bestehen dabei aus der Klasse selbst sowie ihren Methoden- und At-

[4] Techniken zur automatischen Erzeugung eines solchen Vokabulars für eine gegebene Dokumentmenge werden zum Beispiel in [Cro90] diskutiert.

tributdefinitionen. Die Select-Anweisung liefert als Ergebnis Paare aus einem Ähnlichkeitswert in Relation zum gegebenen Anfragetext und einer Objektreferenz für die Klasse. Durch `sort-` werden diese Paare nach der Ähnlichkeit absteigend sortiert. Der Befehl `head[25]` reduziert das Ergebnis schließlich auf die 25 „ähnlichsten" Klassen (vgl. hierzu auch [Hen96b]).

Um das von PCTE verwendete Konzept der Linkkategorien auch bei der Definition der Komponenten, die ein Dokument bilden sollen, ausschöpfen zu können, kann man sich innerhalb eines regulären Pfadausdrucks auch auf die Linkkategorien beziehen. Wollten wir z.B. bei der Berechnung der Beschreibungsvektoren für die Klassen jeweils alle über Links der Kategorie *composition* erreichbaren Objekte mit Ausnahme der Subklassen betrachten, so könnten wir dies durch '`D_vector ([{c shield hat_Subklasse}]*/->.)`' ausdrücken. Das Konstrukt '`{c shield hat_Subklasse}`' adressiert dabei alle Links der Kategorie *composition* mit Ausnahme der Links des Typs *hat_Subklasse*. Durch den Einschluß dieser Linkdefinition in das Iterationskonstrukt '`[...]*`' wird definiert, daß Pfade verfolgt werden sollen, die aus keinem, einem oder mehreren Links bestehen, die der Linkdefinition entsprechen.

2.6. Zugriffsstrukturen und Anfragebearbeitung

Da die beschriebene Anfragesprache in wesentlichen Punkten — mächtige reguläre Pfadausdrücke, Integration von Dokumenten-Retrieval-Funktionalität — über die von herkömmlichen Anfragesprachen gebotene Funktionalität hinausgeht, konnte zu ihrer effizienten Implementierung nur begrenzt auf die im Datenbankbereich bekannten Konzepte zurückgegriffen werden.

Durch die Integration des Vektorraummodells in die Anfragesprache und die damit mögliche Kombination einer Ähnlichkeitssuche mit Bedingungen über Standardattributen erscheint im betrachteten Umfeld der Einsatz mehrdimensionaler Zugriffsstrukturen vielversprechend. Die Idee ist, daß einzelne Dimensionen der Zugriffsstruktur „Standardattributen" wie z.B. einem Namen oder der Anzahl ausgehender Links eines bestimmten Typs entsprechen. Eine weitere logische Dimension kann dann durch einen Beschreibungsvektor definiert sein. Eine Indexdefinition könnte damit z.B. wie folgt aussehen:

```
Index for Klasse:
   1. Name
   2. count _.hat_Methode->.
   3. D_vector (., _.hat_Attr/->., _.hat_Methode/->.)
```

Die erste Dimension entspricht dabei dem Namen der Klasse, die zweite der Anzahl der Methoden und die dritte dem Beschreibungsvektor für das durch den gegebenen Pfadausdruck definierte Dokument (physisch handelt es sich dabei um die Dimensionen 3 bis $3+t$ wenn das Vokabular t Begriffe enthält).

Durch eine Kombination der in [HM95] für Multiattribut-Zugriffsstrukturen beschriebenen Techniken mit den in [Hen96a] dargestellten Techniken zur Bearbeitung von Ähnlichkeitssuchen im Vektorraum können mit einer solchen Zugriffsstruktur Anfragen effizient bearbeitet werden, die eine Ähnlichkeitssuche im Vektorraum mit Bedingungen über Standardattributen kombinieren. Dabei wird die Ähnlichkeitssuche dadurch beschleunigt, daß Teile der Zugriffsstruktur, für die die Bedingungen über den Standardattributen nicht erfüllt sein können, aus der Betrachtung ausgeschlossen werden.

3. Anwendungsbeispiele

Im folgenden wollen wir exemplarisch vier Werkzeuge vorstellen, die mit Hilfe der beschriebenen Retrieval-Dienste implementiert wurden.

3.1. Report-Generator

Im Rahmen eines Software-Entwicklungsprojekts ist es häufig erforderlich, Informationen in Form von Reports aufzubereiten. Die Informationen für einen solchen Report lassen sich i. allg. relativ leicht mit Hilfe einer P-OQL-Anfrage ermitteln. Wir haben daher einen Report-Generator entwickelt, der dem Anwender Reports anbietet, die durch eine Kombination aus einer Anfrage und einer Layout-Beschreibung definiert werden.

Abbildung 3 zeigt exemplarisch eine solche Report-Definition. Die erste Zeile definiert den Namen des Reports und den adressierten Objekttyp (*OOA_Diagramm*). Ferner ist hier spezifiziert, daß der Name des OOA-Diagramms, für das der Report erstellt werden soll, als Parameter in die Report-Definition eingesetzt werden muß. D.h. der konkrete Name wird immer dann, wenn ein Report des Typs *„Klassenaufstellung“* angefordert wird, vom Report-Generator erfragt und in der Report-Definition überall dort ersetzt, wo '#$1#' steht. Den nächsten Teil der Report-Definition bildet die P-OQL-Anfrage, mit der die Informationen für den Report gesucht werden. Im Beispiel handelt es sich um eine sortierte Liste der Klassen mit ihren Attributen. Die Layout-Beschreibung

```
Report "Klassenaufstellung" for OOA_Diagramm, Parameter: Name
Query:  sort+ (select K:Name, K:_.hat_Attr/->Name
                 from D in OOA_Diagramm, K in (D:_.enthaelt/->.)
                 where D:Name = "#$1#")
Layout(LaTeX):  \documentstyle{article}
                \begin{document}
                #BEGIN_ITERATE .#
                  \newpage
                  {\bf Klasse: #.e.1.#}

                  {\em Attribute:}
                  \begin{itemize}
                  #BEGIN_ITERATE .e.2.#
                    \item #.e.2.e.#
                  #END_ITERATE .e.2.#
                  \end{itemize}
                #END_ITERATE .#
                \end{document}
```

Abbildung 3: Beispiel für eine Report-Definition

schließt die Report-Definition ab. Im Beispiel verwenden wir hierzu die LaTeX-Syntax. Die Layout-Beschreibung enthält in '#'-Zeichen eingeschlossene Steueranweisungen, die definieren, wie das Ergebnis der Anfrage in den Report eingearbeitet werden soll. Soll eine bestimmte Folge von Formatierungsanweisungen z.B. für jedes Element einer Liste wiederholt werden, so sind die entsprechenden Formatierungsanweisungen mittels '#BEGIN_ITERATE <Value-*Zugriff*>#' und '#END_ITERATE <Value-*Zugriff*>#' zu klammern. Auf die einzelnen Komponenten des Anfrageergebnisses wird durch Komponentendefinitionen zugegriffen. So greift man mit 'e.' auf ein Element einer Menge, Multimenge oder Liste zu. Mit '2.' wird auf die zweite Komponente einer Struktur zugegriffen.

3.2. Schätzung der Software-Entwicklungskosten

Zur Schätzung des Aufwands, der für eine geplante Software-Entwicklung erforderlich sein wird, sind zahlreiche Verfahren vorgestellt worden. Besonders auf frühen Analyseergebnissen basierende Verfahren, wie die *Object-Point-Methode* [Sne96], bieten sich dabei für eine Repository-basierte Implementierung an. Bei der Object-Point-Methode wird die Schätzung in drei Teilschritten

durchgeführt, in denen je eine Kenngröße für den *Codieraufwand*, den *Aufwand für die Klassenintegration und den Integrationstest* und den *Aufwand für den Systemtest* bestimmt wird. Insgesamt ergeben sich folgende Formeln:

$$
\begin{aligned}
\text{Class-Points} &= (\text{Attribute} + \text{Relationen} \times 2 + \text{Methoden} \times 3) \times \text{Neuheit} \\
\text{Message-Points} &= (\text{Parameter} + \text{Quellen} \times 2 + \text{Ziele} \times 2) \times \text{Komplexität} \times \text{Neuheit} \\
\text{Process-Points} &= (\text{Prozeßtyp} + \text{Varianten}) \times \text{Komplexität} \\
\text{Object-Points} &= \text{Class-Points} + \text{Message-Points} + \text{Process-Points}
\end{aligned}
$$

Die so gewonnene Ausgangsschätzung wird dann noch durch Korrekturfaktoren adjustiert, die Qualitätsanforderungen (QF) — wie Zuverlässigkeit oder Portabilität — und Projekt-Einflußfaktoren (EF) — wie die eingesetzten Methoden — berücksichtigen:

$$\text{Adjusted-Object-Points} = \text{Object-Points} \times \text{QF} \times \text{EF}$$

Ein Adjusted-Object-Point entspricht dann laut Sneed ungefähr einem Entwicklungsaufwand von 0,25 Personentagen.

Wenn nun entsprechende Dokumente zur Klassendefinition, zum Nachrichtenaustausch und zur Prozeßmodellierung im Repository abgelegt sind, dann können die Object-Points direkt mit einer Anfrage aus dem Repository gewonnen werden. Abbildung 4 zeigt eine solche Anfrage, die ein entsprechendes Schema unterstellt und zur Umwandlung von Aufzählungstypen in numerische Werte auf die in P-OQL verfügbare `case`-Anweisung zurückgreift. Diese Anfrage bestimmt die Object-Points für ein Projekt namens *„Abrechnung"*.

Zur Berechnung der Class-Points, Message-Points und Process-Points wird dabei jeweils nach dem gleichen Muster vorgegangen, das wir für die Message-Points kurz verdeutlichen wollen. Ausgangspunkt der Anfrage sind die Projekte (Basismenge `P`). In der `where`-Klausel wird aus diesen Projekten das Projekt mit Namen *„Abrechnung"* selektiert. In der zweiten Basismenge `N` werden alle vom aktuellen Projekt aus erreichbaren Komponenten adressiert. Diese werden in der `where`-Klausel auf Nachrichten eingeschränkt. Für jede Nachricht wird mit Hilfe des `count`-Operators die Anzahl der ausgehenden Links der Typen *hat_Parameter*, *Quelle* und *Ziel* ermittelt.

Die Anfrage aus Abbildung 4 bildet auch die Grundlage für das von uns entwickelte Werkzeug zur Aufwandsschätzung nach der Object-Point-Methode (vgl. hierzu auch [Hen97]). Dieses Werkzeug geht insgesamt wie folgt vor: Zunächst wird der Name des zu betrachtenden Projekts erfragt. Die Berechnung der Object-Points erfolgt dann analog zu der in Abbildung 4 gegebenen

```
(sum                                          // summiere die Class-Points
   (select ((count K:_.hat_Attr->.
             + ((2 * count K:_.Relation->.)
                + (3 * count K:_.hat_Methode->.))) * K:Neuheit)
      from P in Projekt, K in (P:[{c}]+/->.)
      where P:Name = "Abrechnung" and K:. is of type Klasse)
 + (sum                                       // summiere die Message-Points
      (select ((count N:_.hat_Parameter->.
                + ((2 * count N:_.Quelle->.)
                   + (2 * count N:_.Ziel->.)))
                * (N:Neuheit
                   * (case N:Komplexitaet = NIEDRIG ==> +0.75,
                           N:Komplexitaet = MITTEL  ==> +1.00,
                           N:Komplexitaet = HOCH    ==> +1.25)))
         from P in Projekt, N in (P:[{c}]+/->.)
         where P:Name = "Abrechnung" and N:. is of type Nachricht)
  + sum                                       // summiere die Process-Points
      (select (((case Pz:Prozesstyp = SYSTEM   ==> 6,
                      Pz:Prozesstyp = BATCH    ==> 2,
                      Pz:Prozesstyp = ONLINE   ==> 4
                      Pz:Prozesstyp = REALTIME ==> 8) + Pz:Varianten)
                * (case Pz:Komplexitaet = NIEDRIG ==> +0.75,
                        Pz:Komplexitaet = MITTEL  ==> +1.00,
                        Pz:Komplexitaet = HOCH    ==> +1.25))
         from P in Projekt, Pz in (P:[{c}]+/->.)
         where P:Name = "Abrechnung" and Pz:. is of type Prozess)))
```

Abbildung 4: P-OQL-Anfrage zur Bestimmung der Object-Points

Anfrage. Das Ergebnis der Anfrage wird in einem entsprechenden Fenster dargestellt. Der Benutzer kann dann die Korrekturfaktoren angeben. Im Anschluß werden die Adjusted-Object-Points berechnet und angezeigt.

3.3. Dokumenten-Retrieval

Um die Dokumenten-Retrieval-Funktionalität von P-OQL auch dem Nutzer der Software-Entwicklungsumgebung zugänglich zu machen, haben wir ein einfaches Frontend entworfen, mit dem *„Dokumente"* inhaltsbasiert oder zeichenkettenorientiert gesucht werden können. Abbildung 5 zeigt dieses Werkzeug.

Im ersten Teilfenster des Suchwerkzeugs (*Addressed Entity*) kann der Benutzer den als *„Dokumenttyp"* anzusprechenden Objekttyp auswählen. Er kann dabei

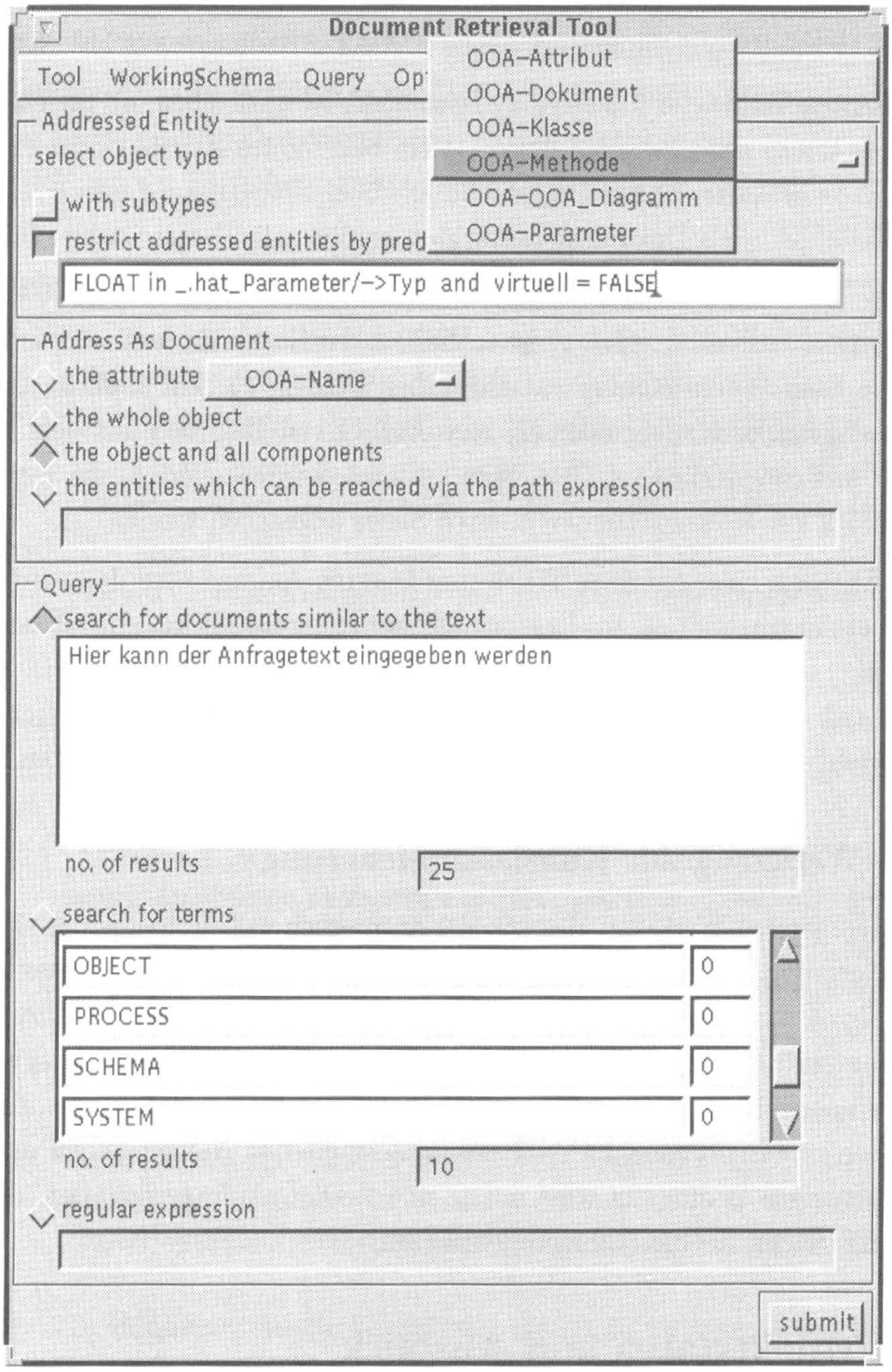

Abbildung 5: Ein Werkzeug zur Suche nach *„Dokumenten“*

angeben, ob er nur an Objekten des Typs selbst oder auch an Subtypen interessiert ist. Zusätzlich kann hier ggf. noch ein einschränkendes Prädikat angegeben werden. Im Beispiel wird dieses Prädikat genutzt, um die Suche auf nichtvir-

tuelle Methoden mit mindestens einem FLOAT-Attribut einzuschränken.

Im zweiten Teilfenster (*Address as Document*) ist anzugeben, ob für ein konkretes Objekt des gewählten Typs (1) ein bestimmtes Attribut oder (2) das Objekt selbst oder (3) das Objekt und alle über *composition*-Links erreichbaren Teilobjekte als Dokument betrachtet werden soll. Ferner ist hier für den erfahrenen Nutzer auch die Eingabe eines regulären Pfadausdrucks möglich.

Im dritten Teilfenster (*Query*) ist schließlich das Suchkriterium zu definieren. Hierzu kann (1) ein Anfragetext angegeben werden, zu dem ähnliche *„Dokumente"* gesucht werden, oder (2) eine Anzahl von Begriffen aus dem Vokabular mit entsprechenden Gewichten versehen werden oder (3) ein regulärer Ausdruck zur zeichenkettenorientierten Suche angegeben werden.

Das Ergebnis einer auf diese Weise spezifizierten Anfrage wird dann zunächst als Liste in einem Standard-Ergebnis-Viewer angezeigt. Wird ein *„Dokument"* (Objekt) aus der Ergebnisliste angewählt, so wird mit Hilfe der Meta-Werkzeugbank ermittelt, welche Werkzeuge zu diesem Objekttyp existieren. Die Werkzeuge werden dem Benutzer ggf. in einer Auswahlleiste angeboten.

3.4. Werkzeug zur Konsistenzsicherung

Bei dem letzten Werkzeug, das wir hier ansprechen wollen, handelt es sich um ein Werkzeug zur Konsistenzsicherung (vgl. hierzu auch [HD96]). Dieses Werkzeug kann aus anderen Werkzeugen (primär aus Editoren) gestartet werden. Es überprüft für das aktuell bearbeitete Dokument (z.B. ein OOA-Diagramm) vordefinierte Bedingungen. Eine solche Bedingung kann z.B. festlegen, daß die Klassen innerhalb eines OOA-Diagramms eindeutige Namen haben müssen. Jede einzelne Bedingung wird durch eine P-OQL Anfrage definiert, die die Objekte sucht, die die Bedingung verletzen.

4. Projektstatus und Ausblick

Die in diesem Papier vorgestellte Funktionalität der Retrieval-Dienste (incl. der Zugriffsstrukturen) ist auf H-PCTE realisiert worden und in weiten Teilen über FTP (`ftp://ftp.informatik.uni-siegen.de/pub/pi/hpcte/`) verfügbar.

Damit existiert eine fundierte Ausgangsbasis für weiterführende Arbeiten in dem beschriebenen Bereich. Gedacht ist dabei derzeit insbesondere (1) an die Weiterentwicklung der Dokumenten-Retrieval-Funktionalität im Hinblick auf

nichttextuelle Aspekte der Ähnlichkeit, (2) an die Entwicklung eines umfassenden Werkzeugs zur Unterstützung des Projektmanagements, (3) an ein Werkzeug zur Suche nach wiederverwendbaren Komponenten, sowie (4) an die Weiterentwicklung der Zugriffsstrukturen und Optimierungstechniken.

Literatur

[Bir95] B. Bird. An open systems SEE query language. In *Proc. 7th Conf. on Software Engineering Environments*, S. 34–47, Noordwijkerhout, 1995.

[BV93] J. Van den Bussche und G. Vossen. An extension of path expressions to simplify navigation in object-oriented queries. In *Deductive and Object-Oriented Databases*, volume 760 of *LNiCS*, S. 267–282, Phoenix, 1993.

[CACS94] V. Christophides, S. Abiteboul, S. Cluet und M. Scholl. From structured documents to novel query facilities. In *Proc. ACM SIGMOD Intl. Conf. on Management of Data*, S. 313–324, Minneapolis, 1994.

[Cat93] R. Cattell, Hrsg. *The Object Database Standard: ODMG-93.* Morgan Kaufmann, San Mateo, 1993.

[CGMB95] C. Clifton, H. Garcia-Molina und D. Bloom. Hyperfile: A data and query model for documents. *VLDB Journal*, 4(1):45–86, 1995.

[Cro90] J.C. Crouch. An approach to the automatic construction of global thesauri. *Information Processing and Management*, 26(5):629–640, 1990.

[FLU94] J. Frohn, G. Lausen und H. Uphoff. Access to objects by path expressions and rules. In *Proc. 20th Intl. Conf. on VLDB*, S. 273–284, 1994.

[Fuh93] N. Fuhr. A probabilistic relational model for the integration of IR and databases. In *Proc. 16th Annual Intl. ACM SIGIR Conf.*, S. 309–317, Pittsburgh, 1993.

[HD96] A. Henrich und D. Däberitz. Using a query language to state consistency constraints for repositories. In *Proc. 7th Intl. Conf. on Database and Expert Systems Applications*, volume 1134 of *LNiCS*, S. 59–68, Zürich, 1996.

[Hen91] S. Henninger. Retrieving software objects in an example-based programming environment. In *Proc. 14th Annual Intl. ACM SIGIR Conf.*, S. 251–260, Chicago, 1991.

[Hen95] A. Henrich. P-OQL: an OQL-Oriented Query Language for PCTE. In *Proc. 7th Conf. on Software Engineering Environments*, S. 48–60, Noordwijkerhout, 1995.

[Hen96a] A. Henrich. Adapting a spatial access structure for document representations in vector space. In *Proc. 5th Intl. Conf. on Information and Knowledge Management*, S. 19–26, Rockville, 1996.

[Hen96b] A. Henrich. Document retrieval facilities for repository-based system development environments. In *Proc. 19th Annual Intl. ACM SIGIR Conf.*, S. 101–109, Zürich, 1996.

[Hen97] A. Henrich. Repository Based Software Cost Estimation. In *Proc. 8th Intl. Conf. on Database and Expert Systems Applications, LNiCS*, Toulouse, September 1997.

[HM95] A. Henrich und J. Möller. Die Nutzung mehrdimensionaler Zugriffsstrukturen für Anfragen über Standardattributen. In *Tagungsband zur GI-Fachtagung „Datenbanksysteme für Büro, Technik und Wissenschaft"*, Informatik Aktuell, S. 212–231, Dresden, 1995. Springer.

[Kel92] U. Kelter. H-PCTE: A high-performance object management system for system development environments. In *Proc. 16th Annual Intl. Computer Software and Applications Conf.*, S. 45–50, Chicago, 1992.

[Kel93a] U. Kelter. An information retrieval common service based on H-PCTE. In *Proc. 6th Conf. on Software Engineering Environments*, S. 101–108, Reading, UK, 1993.

[KS95] D. Konopnicki und O. Shmueli. W3QS: A query system for the world-wide web. In *Proc. 21th Intl. Conf. on VLDB*, S. 54–65, Zürich, 1995.

[MBK91] Y.S. Maarek, D.M. Berry und G.E. Kaiser. An information retrieval approach for automatically constructing software libraries. *IEEE, Transactions on Software Engineering*, 17(8):800–813, 1991.

[Nag93] M. Nagl. Software-Entwicklungsumgebungen: Einordnung und zukünftige Entwicklungslinien. *Informatik-Spektrum*, 16(5):273–280, 1993.

[PCT94] Portable Common Tool Environment - Abstract Specification / C Bindings. ISO IS 13719-1/-2, 1994 sowie ECMA-149/-158, 1993.

[Sci94] E. Sciore. Query abbreviation in the entity-relationship data model. *Information Systems*, 19(6):491–511, 1994.

[Sne96] H.M. Sneed. Schätzung der Entwicklungskosten von objektorientierter Software. *Informatik-Spektrum*, 19(3):133–140, 1996.

[SWY75] G. Salton, A. Wong und C.S. Yang. A vector space model for automatic indexing. *Communications of the ACM*, 18(11):613–620, 1975.

[TTB+90] M. Tedjini, I. Thomas, G. Benoliel, F. Gallo und 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, S. 238–248, 1990.

[VAB96] M. Volz, K. Aberer und K. Böhm. Applying a flexible OODBMS-IRS-coupling for structured document handling. In *Proc. 12th Intl. Conf. on Data Engineering*, S. 10–19, New Orleans, 1996.