

University of Crete
Computer Science Department

Efficient Faceted Exploration Services for Big
Volumes of Information

Nikos Armenatzoglou
Master's Thesis

Heraklion, 26 March 2010

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΡΗΤΗΣ
ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΚΑΙ ΤΕΧΝΟΛΟΓΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΕΠΙΣΤΗΜΗΣ ΥΠΟΛΟΓΙΣΤΩΝ

Αποδοτική Υποστήριξη Λειτουργιών Πολυδιάστατης
Πλοήγησης σε Μεγάλους Όγκους Πληροφοριών

Εργασία που υποβλήθηκε από τον
Νικόλαο Α. Αρμενατζόγλου
ως μερική εκπλήρωση των απαιτήσεων για την απόκτηση
ΜΕΤΑΠΤΥΧΙΑΚΟΥ ΔΙΠΛΩΜΑΤΟΣ ΕΙΔΙΚΕΥΣΗΣ

Συγγραφέας:

Νικόλαος Αρμενατζόγλου, Τμήμα Επιστήμης Υπολογιστών

Εισηγητική Επιτροπή:

Ιωάννης Τζιτζικας, Επίκουρος Καθηγητής, Επόπτης

Δημήτρης Πλεξουσάκης, Καθηγητής, Μέλος

Γρηγόρης Αντωνίου, Καθηγητής, Μέλος

Δεκτή:

Πάνος Τραχανιάς, Καθηγητής
Πρόεδρος Επιτροπής Μεταπτυχιακών Σπουδών
Ηράκλειο, 26 Μαρτίου 2010

Efficient Faceted Exploration Services for Big Volumes of Information

Nick Armenatzoglou

Master's Thesis

Computer Science Department, University of Crete

Abstract

Although most information needs have exploratory nature, current web search engines do not support these needs sufficiently. The objective of this thesis is the development of effective and efficient services for fulfilling such needs. Specifically, this thesis proposes the extension of the "traditional" interaction scheme (query-and-ranked answer) with exploration services based on dynamic faceted taxonomies.

The advantages of this interaction scheme is that it bridges the gap between querying and browsing, it provides an overview of the information space and gives the user the ability to reduce in a flexible and efficient manner the size of the information space according to his interests.

To support this model over a Web Search Engine (WSE), we studied various architectures and implementation approaches and subsequently we designed and implemented a general purpose API and a Web-based GUI interface. Furthermore, we examined several methods for exploiting the metadata that are available at the index of a WSE plus those that can be dynamically generated (e.g. from applying real-time clustering results techniques or other data mining techniques).

The above techniques have been applied and evaluated over the Mitos WSE, making it the first WSE that supports this model. The performance of the proposed techniques was evaluated experimentally, while various variants of the model (regarding the choice of facets) were evaluated by users comparatively. The evaluation showed that users can directly use this model of interaction (no need for training) and that they are quite effective.

Finally, the thesis proposes and analyzes an advanced method for scaling up these services, so that to enable real-time exploration of billion-sized information spaces. More precisely, it proposes an approach that allows computing the zoom points without having to perform any operation on the object-base at browsing time. The proposed technique is based on CTCA (Compound Term Composition Algebra), special indices and algorithms. The above techniques have been applied to a range of large data sets and we compare it with the existing techniques. The analytic study showed that current methods cannot scale to billion-sized collections. The experimental evaluation showed that the proposed approach is efficient enough for such magnitudes.

Supervisor: Yannis Tzitzikas
Assistant Professor

Αποδοτική Υποστήριξη Λειτουργιών Πολυδιάστατης Πλοήγησης σε Μεγάλους Όγκους Πληροφοριών

Νίκος Αρμενατζόγλου

Μεταπτυχιακή Εργασία

Τμήμα Επιστήμης Υπολογιστών, Πανεπιστήμιο Κρήτης

Περίληψη

Αν και οι περισσότερες πληροφοριακές ανάγκες έχουν εξερευνητικό χαρακτήρα, οι σημερινές μηχανές αναζήτησης του Ιστού δεν τις καλύπτουν επαρκώς. Στόχος της εργασίας είναι η ανάπτυξη αποτελεσματικών και αποδοτικών υπηρεσιών για την ικανοποίηση τέτοιων αναγκών. Συγκεκριμένα προτείνεται η επέκταση του καθιερωμένου τρόπου αλληλεπίδρασης (υποβολή της επερώτησης και επιστροφή διαβαθμισμένων αποτελεσμάτων) με υπηρεσίες εξερευνησης βασισμένες στο υπόδειγμα των πολυδιάστατων δυναμικών ταξινομιών.

Τα πλεονεκτήματα αυτού του μοντέλου αλληλεπίδρασης είναι ότι γεφυρώνει το χάσμα μεταξύ της πρόσβασης μέσω ερωτήσεων και πρόσβασης μέσω πλοήγησης, παρέχει εποπτεία του πληροφοριακού και επιτρέπει στο χρήστη να μειώσει γρήγορα και εύχρηστα τον πληροφοριακό χώρο ενδιαφέροντος.

Για την υποστήριξη αυτού του μοντέλου στην αναζήτηση στον ιστό, μελετήθηκαν και αξιολογήθηκαν διάφορες αρχιτεκτονικές και τρόποι υλοποίησης και κατόπιν σχεδιάστηκε και υλοποιήθηκε ένα API, και μια διαδικτυοκεντρική γραφική διεπαφή γενικού σκοπού. Επίσης μελετήθηκαν οι τρόποι αξιοποίησης των μεταδεδομένων που διαθέτει το ευρετήριο μιας μηχανής αναζήτησης, καθώς και των δυναμικά παραγόμενων μεταδεδομένων (π.χ. ως αποτέλεσμα τεχνικών ομαδοποίησης πραγματικού χρόνου ή άλλων τεχνικών εξόρυξης πληροφορίας).

Τα παραπάνω εφαρμόστηκαν και αξιολογήθηκαν στη μηχανή αναζήτησης Μίτος καθιστώντας την, την πρώτη μηχανή αναζήτησης που υποστηρίζει αυτό το μοντέλο. Οι επιδόσεις αξιολογήθηκαν πειραματικά, ενώ διάφορες παραλλαγές του μοντέλου (κυρίως ως προς την επιλογή των διαστάσεων πλοήγησης) αξιολογήθηκαν συγκριτικά από χρήστες. Η αξιολόγηση κατέδειξε ότι ένας απλός χρήστης της μηχανής αναζήτησης εύκολα μπορεί να χρησιμοποιεί τη συγκεκριμένη λειτουργικότητα και να φτάσει στο επιθυμητό αποτέλεσμα.

Τέλος, προτείνονται και αναλύονται προηγμένοι τρόποι κλιμάκωσης των υπηρεσιών ώστε να καταστεί εφικτή η πλοήγηση σε πραγματικό χρόνο απαντήσεων της τάξης του 10^9 . Συγκεκριμένα, προτείνεται ο υπολογισμός των σημείων εστιασμού με μια προσέγγιση η οποία δεν απαιτεί υπολογισμούς επί των ευρετηριασμένων αντικειμένων κατά τη διάρκεια της εξερεύνησης. Η προσέγγιση αυτή βασίζεται στην άλγεβρα CTCA και σε ειδικά σχεδιασμένα ευρετήρια και αλγορίθμους. Οι παραπάνω τεχνικές εφαρμόστηκαν σε μεγάλους όγκους πληροφοριών και συγκρίθηκαν με τις υπάρχουσες τεχνικές. Η αναλυτική μελέτη έδειξε ότι καμιά από τις υπάρχουσες τεχνικές δεν μπορεί να εφαρμοστεί σε μεγάλους όγκους πληροφοριών. Η πειραματική αξιολόγηση έδειξε ότι η συγκεκριμένη προσέγγιση είναι αρκετά αποδοτική για τέτοιου μεγέθους όγκου πληροφοριών.

Επόπτης Καθηγητής: Γιάννης Τζιτζικας

Επίκουρος Καθηγητής

Ευχαριστίες

Σε αυτό το σημείο θα ήθελα να ευχαριστήσω θερμά τον επόπτη μου επίκουρο καθηγητή κ. Ιωάννη Τζίτζικα για τη πολύ καλή συνεργασία μας καθώς και για την ουσιαστική του καθοδήγηση και συμβολή στην ολοκλήρωση αυτής της εργασίας. Μέσα από αυτή τη συνεργασία κέρδισα πάρα πολλά και σε διαφορετικά επίπεδα. Όποτε ακολούθησα τις συμβουλές και το συστηματικό τρόπο δουλείας του είχα πολύ καλά αποτελέσματα. Επίσης, θα ήθελα να τον ευχαριστήσω για όλες τις ευκαιρίες που μου έδωσε μέσω του Πανεπιστημίου Κρήτης και του Ινστιτούτου Πληροφορικής του Ιδρύματος Τεχνολογίας και Έρευνας.

Επιπλέον, θα ήθελα να ευχαριστήσω τους καθηγητές κ. Δημήτρη Πλεξουσάκη και κ. Γρηγόρη Αντωνίου για τη προθυμία τους να συμμετάσχουν στην εξεταστική επιτροπή της μεταπτυχιακής μου εργασίας καθώς και για τις εύστοχες παρατηρήσεις τους.

Θα ήταν παράληψη μου να μην ευχαριστήσω τους φίλους και συμφοιτητές μου που έκαναν αυτό το διάστημα πολύ ευχάριστο (μερικές φορές με τη βοήθεια ρακής/ουίσκι). Ιδιαίτερα θα ήθελα να ευχαριστήσω το Παναγιώτη, το Γιάννη και το Νίκο για το χρόνο και τις παρατηρήσεις τους στη προετοιμασία της παρουσίασης.

Τέλος, θα ήθελα να ευχαριστήσω τους γονείς μου Ανέστη και Μαρία και την αδερφή μου Μαρία (ναι, έχουν το ίδιο όνομα) για την αγάπη τους.

Contents

Table of Contents	v
List of Figures	xi
1 Introduction	1
1.1 Background - History	3
1.1.1 Objects' Indexing	3
1.1.2 FDT	4
1.2 Advantages of Dynamic Taxonomies	5
1.3 Contributions	7
1.4 Organization of the thesis	7
2 Faceted Exploration Model	9
2.1 A Model for Facet-based Exploration	9
2.1.1 Top Element	10
2.1.2 Zoom-in	11
2.1.3 Zoom-out	12
2.1.4 Zoom-Side	13
2.1.5 Presentation and Ranking of Zoom-in points	14
2.1.6 Restriction of a Materialized Faceted Taxonomy	14
2.1.7 Synopsis	16
2.2 Related Approaches	16

3	Architectures and Related Work	19
3.1	FDT Interaction & Computational Requirements	19
3.1.1	A State-based Interaction Method	20
3.1.1.1	States	20
3.1.1.2	State Visualization	20
3.1.1.3	State Transition	21
3.1.2	General Evaluation Approaches	22
3.2	Data Structures & Algorithms	23
3.2.1	Notations	23
3.2.2	Storage Policies	24
3.2.2.1	Data Structures	26
3.2.3	Algorithms and Complexity	26
3.2.3.1	EV_i & V_i Computation	26
3.2.3.2	Zoom-out points Computation	29
3.2.3.3	Count Information	30
3.2.3.4	Conclusions of the Analysis	30
3.3	Possible Architectures	32
3.3.1	(MEM) Architecture	32
3.3.2	(DB) Architecture	34
3.3.2.1	From the Relational to the Faceted Data Model: Method- ological Comments	35
3.3.2.2	On SQL implementation	37
3.3.2.3	Direct and Indirect Narrower/Broader terms of a term . .	38
3.3.2.4	Maximal Incomparable Terms of a Term	39
3.3.2.5	Direct and Indirect Narrower/Broader terms of a set of terms	39
3.3.2.6	Model Interpretations	40
3.3.2.7	Object Descriptions	42
3.3.2.8	Complete Descriptions	42
3.3.2.9	V_i & EV_i Computation	44

3.3.2.10	Zoom-out points Computation	46
3.3.2.11	Count Information	47
3.4	Faceted Exploration User Interfaces	47
3.5	FDT in Commercial Web-sites	53
3.5.1	Commercial Faceted Metadata Search Engines	53
3.5.2	XFML	54
4	fileXplorer & Applications	57
4.1	fileXplorer API	57
4.1.1	Specifications	57
4.1.1.1	Class Diagrams	59
4.1.2	An Example of Using the API	60
4.1.3	Desktop-based Client	68
4.1.4	Experimental Evaluation	68
4.2	Application on a Web Search Engine	71
4.2.1	Mitos WSE	71
4.2.2	Exploratory web searching with dynamic taxonomies and results clustering	73
4.2.2.1	Coupling Static and Dynamically-mined Metadata for Ex- ploration	76
4.2.2.2	Incremental Algorithm for Exploration	77
4.2.2.3	Implementation	77
4.2.2.4	Experimental Results	79
4.2.2.5	Evaluation of Usability	81
4.2.3	Exploratory web searching with Entity Mining	85
4.3	EO User Service Next Generation Project (EO USNG)	87
4.4	Experimental Results on DB-R Architecture	87
4.4.1	(DB-R) With No Hierarchically Organized Values	88
4.4.2	(DB-R) With Hierarchically Organized Values	88
4.4.3	(DB-R) With Hierarchically Organized Values (Bigger Data Set) . .	91

5	Extensions For Scalability	93
5.1	A Global-scale Exploration Scenario	93
5.2	Interaction Scheme for Large Collections	95
5.2.1	CTCA-based Approach	97
5.2.1.1	V_i & EV_i Computation	97
5.2.1.2	A short introduction to CTCA	98
5.2.1.3	Compound term validity and CTCA	100
5.2.1.4	Mining a CTCA expression	102
5.2.1.5	Approximating Zoom Point Count Information	102
5.2.1.6	Optimizations	103
5.2.1.7	Related Work on Labeling Schemes	106
5.2.2	TLOI-based Approach	107
5.2.2.1	Indices for Storing the Interpretations	107
5.2.2.2	Indices for storing the taxonomies	109
5.2.2.3	Zoom-in points Computation	110
5.2.2.4	TLOI Advantages	111
5.2.2.5	TLOI on DAG and Multiple Classification	111
5.2.3	Changes over Materialized Faceted Taxonomy	112
5.2.3.1	CTCA Updates	113
5.2.3.2	TLOI Updates	113
5.3	Evaluation	114
5.3.1	Analytical Evaluation	115
5.3.2	Computation of \bar{I} : Time Perspective	116
5.3.3	CTCA Validity Checking	118
5.3.3.1	Estimation of CTCA Parameters Plurality	118
5.3.3.2	isValid Experiments Without Optimizations	119
5.3.3.3	isValid Experiments With Optimizations	120
6	Conclusion and Future Work	123

7	Appendix	131
7.1	Proofs	131

List of Tables

2.1	Basic notions and notations	11
2.2	Interaction notions and notations	17
3.1	Zoom-in points Computation' Complexities	31
3.2	Automatic Hierarchy Creation Examples	37
3.3	Faceted Metadata Search Engines in commercial sites	53
4.1	Table of Symbols	76
4.2	Top- <i>C</i> Integration Timings for non-Incremental and Incremental Algorithms (in seconds)	79
4.3	User Evaluation Tasks	82
4.4	User Evaluation Form	82
4.5	User Satisfaction, Preference and Completeness percentage results per Interface	83
4.6	Number of User Queries and Clicks (as recorded in the log)	83
4.7	User Satisfaction and Preference percentages per Interface (per task)	85
4.8	Partial database schema of Mitos	88
4.9	Database schema of small synthetic dataset	89
4.10	Database schema of large synthetic dataset.	91
5.1	Global Web Scenario	95
5.2	Basic notions and notations	100
5.3	Comparison table according Global Index Scenario	116
5.4	<i>isValid</i> execution times without optimizations	120

List of Figures

1.1	Faceted Taxonomies UI	2
1.2	Parametric Search	5
1.3	FSEs	6
2.1	Example of a Materialized Faceted Taxonomy	10
2.2	Examples of side zoom-in conditions (a) faceted taxonomies, (b) a non-tree taxonomy (i.e. DAG) and (c) multiple classification. With black are the current zoom-in points and with grey the side ones	13
2.3	Example of a Restricted Materialized Faceted Taxonomy	16
3.1	Visualization Modes Example	21
3.2	Extensions Comparison: Simple example	25
3.3	A simple MFT	33
3.4	Storage indices according to [5]	33
3.5	Storage indices according to [38]	35
3.6	Constructing Complete Descriptions	43
3.7	Flamenco User Interface	48
3.8	E-government portal with dynamic taxonomies	49
3.9	/facet User Interface	49
3.10	Museum Finland User Interface	50
3.11	Fathumb User Interface	51
3.12	Veturi User Interface	52
3.13	DBLP User Interface	52
3.14	XFML file example	54

4.1	flexplorer Class Diagram	61
4.2	Terms' Package Class Diagram	62
4.3	Terminologies' Package Class Diagram	63
4.4	Taxonomies' Package Class Diagram	64
4.5	Facets' Package Class Diagram	65
4.6	Faceted Taxonomies' Package Class Diagram	65
4.7	Materialized Faceted Taxonomies' Package Class Diagram	66
4.8	Desktop-based Client: Welcome Screen	69
4.9	Desktop-based Client: Facets and Objects Loading	69
4.10	Desktop-based Client: Faceted Exploration UI	70
4.11	Time to load results to flexplorer	70
4.12	Time to compute zoom-in points	71
4.13	Mitos & flexplorer Sequence Diagram	72
4.14	Mitos user interface: Interpretations, Descriptions	73
4.15	Mitos user interface: Focus	74
4.16	Modified Faceted Exploration UI according user's preferences	74
4.17	Screenshot of Mitos WSE	78
4.18	Steps (a)-(c) of running scenario	80
4.19	Faceted Taxonomies and Entity Mining	86
4.20	ESA-USNG User Interface	87
4.21	Experimental Results on DBMS	88
4.22	Experimental results on synthetic databases.	90
5.1	Product and minus-product operation example	99
5.2	Self-plus-product operation example	99
5.3	Labeling algorithm over the Faceted Taxonomy	104
5.4	Indices for storing \mathcal{P} parameter	105
5.5	Indices for storing \mathcal{N} parameter	106
5.6	DDC labeling algorithm over the Faceted Taxonomy	107
5.7	Store Indices	109
5.8	TLOI vs Object-extended Agrawal's Labeling	110

5.9	TLOI on DAG	112
5.10	TLOI with multiple classification	113
5.11	$\bar{I}(ctx)$ Measures	117
5.12	<i>isValid</i> execution times with terms labeling optimization	121

Chapter 1

Introduction

Nowadays, the size of the available information in digital format is extremely huge. According to Netcraft¹ the Web (until April 2009) contains 232 million web sites and according to [19], the public indexable Web contained 11.5 billion pages on 2005. Additionally, there are a lot of digital libraries which are not yet published to the Internet. This extremely fast rate of the digital information growth is caused by the people which can continuously put information onto the Internet.

Every day, there are millions of users that perform a request to a web search engine or others who browse web catalogues in order to find the information needed. There are two different information access modes: *querying* and *browsing*. We could say that query services are either too simplistic (e.g. free text queries), or too sophisticated (e.g. SQL queries, or Semantic Web Queries). On the other hand, web catalogues which provide browsing services, are either too simplistic (e.g. plain Web links) or very application specific (dynamic pages derived by specific application programs).

Faceted exploration services have recently gained a lot of attention among researchers and have been used in various application domains (for more see [39, 24, 22, 27, 29, 44, 17]. In brief, FSE (Faceted Search Engines) can switch easily between searching and browsing and allow users to see exactly the options that are available at any time for restricting their focus. The aim of a faceted metadata search engine is to provide guided exploration and information thinning services in order to guide the user to reach his goal.

¹http://news.netcraft.com/archives/web_server_survey.html

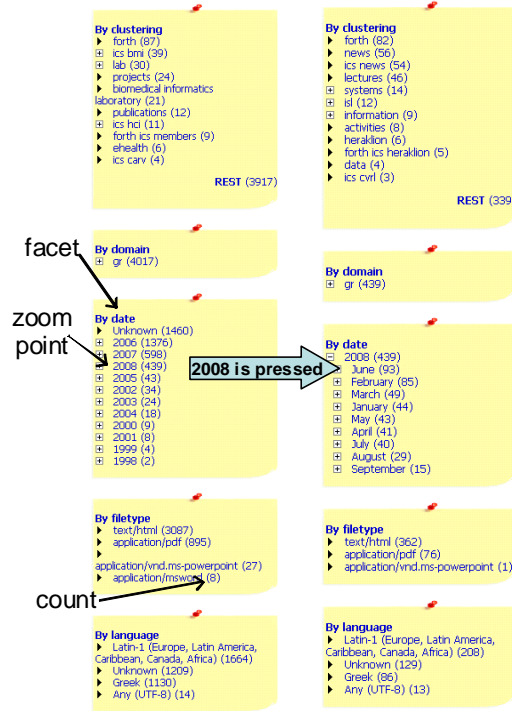


Figure 1.1: Faceted Taxonomies UI

For instance, if we assume that the objects of the domain are indexed by descriptions over a multidimensional space where each dimension is associated with a taxonomy (hierarchy of terms/values), then a FSE that supports the interaction paradigm of *faceted dynamic taxonomies* (FDT), shows only those terms of the taxonomy of each facet that lead to non-empty answer sets, and the user can gradually restrict his focus by clicking on such terms, e.g. see Figure 1.1.

In brief, FSE:

- display the current results in multiple categorization schemes (e.g. based on meta-data terms, such as size, price or date),
- display categories or values (usually called *zoom points*) leading to non-empty results,
- display the count of the indexed objects of each category/value (i.e. the number of results the user will get if he restricts his focus using that category).
- support a session-based dialog in contrast to the state-less query-and-answer dialog of WSE.

On the other hand, exploratory searching poses several open questions and challenges (e.g. see [50]). One critical and open problem [48] is how a search engine could quickly compute (or estimate) the *zoom points* for every result that matches a particular query over a large corpus of documents each possibly described by many facets. All performance measurements that have been reported in related works (e.g. in [59, 43, 5]) are over small collections of objects (10^4 to $8 * 10^5$), and to the best of our knowledge there is not any system or work that attempts to scale such services (i.e. the computation of *zoom points*) for larger collections.

The general objective of this thesis is to study and develop effective and efficient faceted exploration services which can fulfill the needs of a web search engine user for fast and efficient information exploration.

1.1 Background - History

This section presents the background/history of objects' indexing and FDT.

1.1.1 Objects' Indexing

The categorization of our knowledge occupy the mind of the man from the first time that he was characterized as "*homo universalis*"². First of all, Plato (427 BC - 348 BC), in his Statesman dialogue introduces the approach of grouping objects based in their similar properties (classical categorization). His approach was further explored and systematized by one of his best students, Aristotle. Aristotle (384 BC - 322 BC) analyzed the differences between classes and objects and applied Plato's classical categorization scheme to the classification of living beings [36].

According Aristotelian classification, categories are discrete entities and should be mutually exclusive and collectively exhaustive. A category is characterized by a set of properties. Finally, any entity of the given classification belongs to one and only one category. Aristotle's classification contains some elements which still existed in the twenty

²The term "*homo universalis*" (Latin for "universal man" or "man of the world") is used to describe a person who is well educated or who excels in a wide variety of subjects or fields.

century.

Many years later, S. R. Ranganathan (1892 - 1972) developed the first major analytico-synthetic classification system, the Colon classification [37]. Colon classification (CC) is a system of library classification. Its name comes from the use of colons to separate categories (or facets). CC uses five primary categories, or facets to further specify the sorting of a publication collectively called "PMEST": Personality, Matter (or property), Energy, Space, Time.

Research groups (1950 - 1970) simplified Ranganathan's classification scheme: a facet must represent only one characteristic and suppose that an object can be classified by only one term of each facet.

Nowadays, multiple classification (i.e. an object can be indexed with more than one terms from the same facet) is common requirement.

1.1.2 FDT

FDT uses metadata for switching easily between *querying* and *browsing*. According to *Wikipedia* ³, metadata is information about information: more precisely, it is structured information about resources. For example, metadata would document data about data elements or attributes, (name, size, data type, etc) and data about records or data structures (length, fields, columns, etc) and data about data (where it is located, how it is associated, ownership, etc.). Metadata may include descriptive information about the context, quality and condition, or characteristics of the data. So, faceted exploration is an approach to structured data access.

A traditional approach to structured data access is the *parametric search*. Parametric search fits a number of simultaneous criteria (the parameters of the search). For example, finding a house within one of three neighborhoods, \$3-600,000, with at least 3 bedrooms and 2 baths. Figure 1.2 presents two user interfaces of advanced search from commercial sites.

However, the parametric search is so user specific and does not allow the browsing of information space. For this reason, researchers proposed the faceted and dynamic

³<http://www.wikipedia.org>

Figure 1.2: Parametric Search

taxonomies. Nowadays, FDT are used in several application domains e.g. web portals, libraries. Figure 1.3 presents the UIs from three commercial web sites, in (a) the Tower Records ⁴, in (b) the American Express Travel and Leisure ⁵ and in (c) the Beach House ⁶. According to the Knowledge Architecture Professional Services Group (KAPS Group) ⁷, 69% of e-commerce web sites used faceted navigation, 77% used navigation, 6% used faceted classification in search but no browse, 17% had both search and browse and 67% only used single point entry, no progressive filtering, just categories.

1.2 Advantages of Dynamic Taxonomies

Faceted interaction scheme seems to be very quickly understood by end-users as its user interface is very user-friendly, facets are hierarchical organized and the user has only to execute zoom operations (select/deselect a *zoom point*).

Below we enumerate the advantages of dynamic taxonomies as they described in [39]:

- The user is effectively guided to reach his goal: at each stage he has a complete list of all related concepts (i.e. a complete taxonomic summary of his current focus).
- Transparency: the user is in charge and knows exactly what's happening.
- Schema design, where a faceted structure leads to minimal and flexible schemata.

⁴<http://www.towerrecords.com/>

⁵<http://www.travelandleisure.com/index.cfm>

⁶<http://www.beachhouse.com>

⁷<http://www.kapsgroup.com>

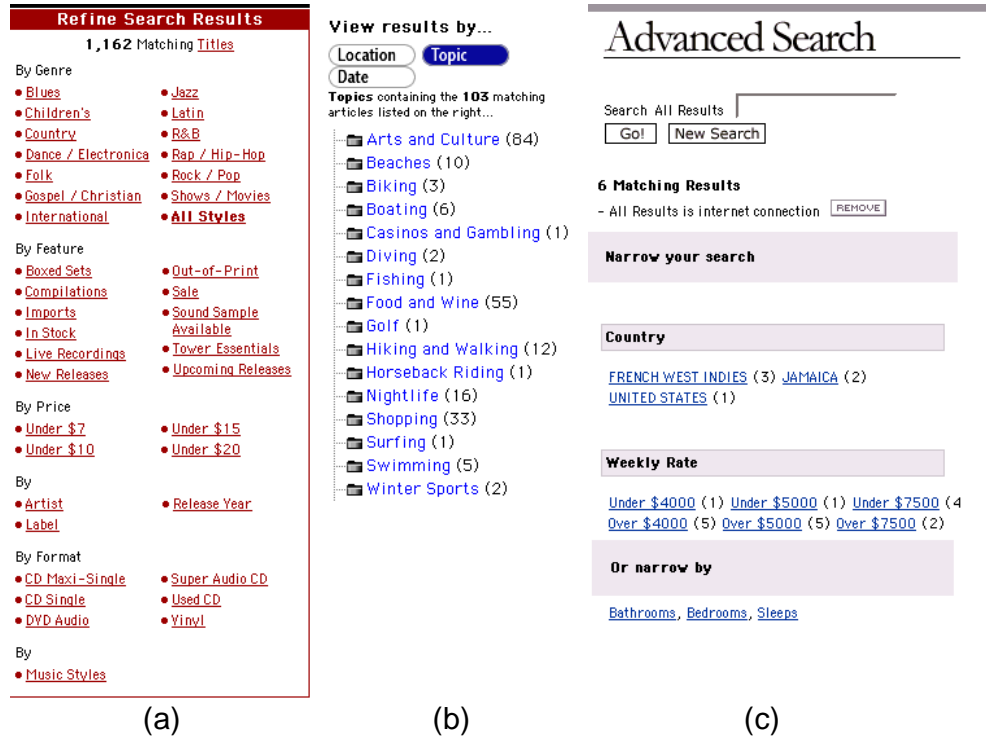


Figure 1.3: FSEs

- Search effectiveness, because dynamic taxonomies have an extremely fast convergence to small result sets.
- Discovery of unexpected relationships between facet's terms e.g. imagine a collection of recipes where the most recipes which are classified under the term "Summer" of the facet "By Season" are also classified under the term "Tomatoes" of the facet "By Ingredients", so we know that the most foods in summer contain tomato.
- No empty results, by construction.
- Any combination of concepts (AND, OR, NOT) is supported.
- Easy to accommodate reviews, popularity, etc.
- Simple integration with other retrieval techniques (IR, DB).

Finally, the main advantage of the faceted exploration interaction scheme is the information thinning after a zoom operation and that user is sure that there is no any better object than the one that he is about to select. Sacco in [39] shows that 3 zoom

operations on terminal concepts are sufficient to reduce a 10,000,000 object information base described by a taxonomy with 1,000 *zoom points* to an average of 10 objects.

1.3 Contributions

The main contributions of this thesis are:

- the analytical study of a number of possible architectures that one can follow to develop a faceted exploration application with respect to the available resources i.e RAM, and the size of the collection,
- the implementation of a Main Memory API written in Java which provides the core functionality for implementing the faceted exploration model,
- the usage of the API in various applications e.g. in Mitos web search engine and European Space Agency web portal,
- the dynamic coupling of results clustering with dynamic faceted taxonomies resulting to an effective, flexible and efficient exploration experience, and
- an investigation of various techniques that could be used for advancing the scalability of such services.

1.4 Organization of the thesis

This thesis is organized as follows:

Chapter 2 introduces a formal model for facet-based exploration services.

Chapter 3 introduces and describes possible architectures for realizing this model and discusses the related work.

Chapter 4 presents an API for supporting faceted exploration and various applications that were implemented with experimental results.

Chapter 5 elaborates on techniques for providing faceted exploration services over terra-sized collections of data.

Chapter 6 summarizes the results of this thesis and identifies topics that are worth further work and research.

Chapter 2

Faceted Exploration Model

Section 2.1 introduces a formal model for facet-based exploration services, while section 2.2 presents related approaches.

2.1 A Model for Facet-based Exploration

This section introduces a formal model aiming at capturing all key notions appearing in [41], [55], and [14]. In brief Obj is a set of objects, \mathcal{T} is a set of terms that may be hierarchically organized, the elements of Obj can be described with respect to one or more aspects (facets), while the description of an object with respect to one facet consists of assigning to the object one or more terms from the taxonomy that corresponds to that facet. Table 2.1 defines formally and introduces notations for *terms*, *terminologies*, *taxonomies*, *faceted taxonomies*, *interpretations*, *descriptions* and *materialized faceted taxonomies* (for details refer to [55, 53]).

An example of a materialized faceted taxonomy, i.e. a faceted taxonomy consisting of four facets and accompanied by a set of object indexes, is shown in Figure 2.1.

Each facet F_i is associated with a name (a String) and a taxonomy. The same taxonomy may be associated with more than one facets (for instance, for indexing flights we may have two facets, named "from" and "to", associated with the same taxonomy "Location"). However, by prefixing the name of each term with the facet name, we may assume that all facet terminologies are disjoint (as stated in Table 2.1).

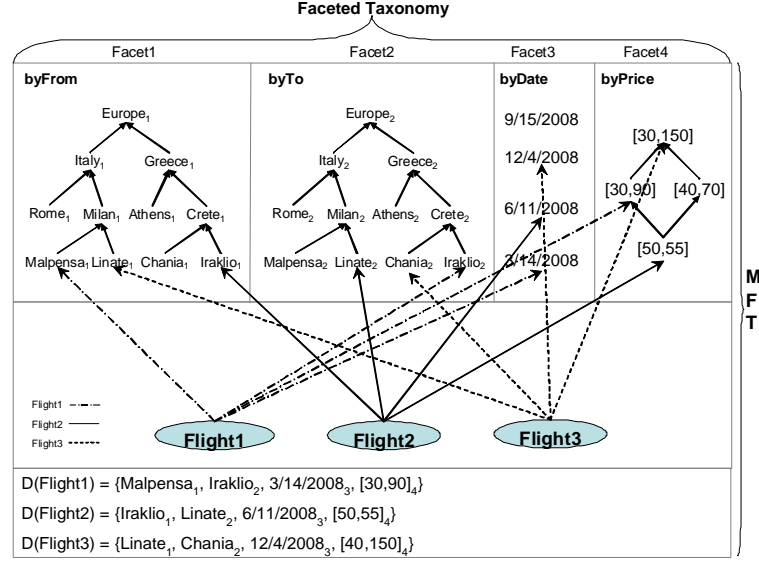


Figure 2.1: Example of a Materialized Faceted Taxonomy

For our purposes, we need to adopt a minimal query language. A query is a compound term s (i.e. a conjunction of terms) and its answer is the set of objects $\bar{I}(s)$ (as defined in Table 2.1). Of course, boolean expressions and more complex query operators can be straightforwardly supported.

As interaction is of prominent importance, now we define formally the notions needed for describing interaction. Any subset of \mathcal{T} is a possible *focus*. For reasons of minimality, we shall hereafter consider foci that are *redundancy free*. A focus ctx (i.e. $ctx \subseteq \mathcal{T}$) is redundancy free if $ctx = \text{minimal}_{\preceq}(ctx)$. For example, $ctx = \{\text{Greece}, \text{Athens}\}$ is not redundancy free because $\text{minimal}_{\preceq}(ctx) = \{\text{Athens}\}$. The *content* of a focus ctx , is the set of objects $\bar{I}(ctx)$. We could also refine this notion and distinguish the *shallow* content $I(ctx)$, from the *deep* content $\bar{I}(ctx)$. In our example, $I(\text{byFrom.Italy}) = \emptyset$, while $\bar{I}(\text{byFrom.Italy}) = \{\text{Flight1}, \text{Flight3}\}$.

2.1.1 Top Element

Each facet i independently to its structure (flat or hierarchically organized) has an unique auxiliary element \top_i , which is the top element of the taxonomy (\mathcal{T}_i, \leq) i.e. $\top_i = \max_{\leq}(\mathcal{T}_i)$.

\top_i is used for keeping the heads of each facet hierarchy. Furthermore, for a focus ctx , if $ctx_i = \emptyset$, then we will assume that $ctx_i = \top_i$. The figures which depict facet hierarchies

MATERIALIZED FACETED TAXONOMIES		
Name	Notation	Definition
<i>terminology</i>	\mathcal{T}	a set of <i>terms</i> (can capture categorical/numeric values)
<i>subsumption</i>	\leq	a partial order (reflexive, transitive and antisymmetric)
<i>taxonomy</i>	(\mathcal{T}, \leq)	\mathcal{T} is a terminology, \leq a subsumption relation over \mathcal{T}
<i>broaders of t</i>	$B^+(t)$	$\{t' \mid t < t'\}$
<i>narrowers of t</i>	$N^+(t)$	$\{t' \mid t' < t\}$
<i>t and its broaders</i>	$B^*(t)$	$\{t\} \cup B^+(t)$
<i>t and its narrowers</i>	$N^*(t)$	$\{t\} \cup N^+(t)$
<i>direct broaders of t</i>	$B(t)$	$\text{minimal}_{<}(B^+(t))$
<i>direct narrowers of t</i>	$N(t)$	$\text{maximal}_{<}(N^+(t))$
<i>faceted taxonomy</i>	$\mathcal{F} = \{F_1, \dots, F_k\}$	$F_i = (\mathcal{T}_i, \leq_i)$, for $i = 1, \dots, k$ and all \mathcal{T}_i are disjoint
<i>compound term over \mathcal{T}</i>	s	any subset of \mathcal{T} (i.e., any element of $\mathcal{P}(\mathcal{T})$)
<i>compound ordering \preceq</i>	$s \preceq s'$	$s \preceq s'$ iff $\forall t' \in s' \exists t \in s$ s.t. $t \leq t'$
<i>broaders of s</i>	$B^+(s)$	$\{s' \in \mathcal{P}(\mathcal{T}) \mid s \prec s'\}$
<i>narrowers of s</i>	$N^+(s)$	$\{s' \in \mathcal{P}(\mathcal{T}) \mid s' \prec s\}$
<i>direct broaders of s</i>	$B(s)$	$\text{minimal}_{\preceq}(B^+(s))$
<i>direct narrowers of s</i>	$N(s)$	$\text{maximal}_{\preceq}(N^+(s))$
<i>object domain</i>	Obj	any denumerable set of objects
<i>interpretation of \mathcal{T}</i>	I	any function $I : \mathcal{T} \rightarrow 2^{Obj}$
<i>materialized faceted taxonomy</i>	(\mathcal{F}, I)	\mathcal{F} is a faceted taxonomy $\{F_1, \dots, F_k\}$, I is an interpretation of $\mathcal{T} = \bigcup_{i=1,k} \mathcal{T}_i$
<i>Top element of facet i</i>	\top_i	$\top_i = \text{maximal}_{\leq}(\mathcal{T}_i)$
<i>ordering of interpretations</i>	$I \sqsubseteq I'$	$I(t) \subseteq I'(t)$ for each $t \in \mathcal{T}$
<i>model of (\mathcal{T}, \leq) induced by I</i>	I	the minimal model that is greater than I $I(t) = \cup \{I(t') \mid t' \leq t\}$
<i>extension of s in I and in \bar{I}</i>	$I(s), \bar{I}(s)$	$I(s) = \cap \{I(t) \mid t \in s\}$ and $\bar{I}(s) = \cap \{\bar{I}(t) \mid t \in s\}$
<i>Description of o wrt I</i>	$D_I(o)$	$D_I(o) = \{t \in \mathcal{T} \mid o \in I(t)\}$
<i>Description of o wrt \bar{I}</i>	$D_{\bar{I}}(o) \equiv \bar{D}_I(o)$	$D_{\bar{I}}(o) = \{t \in \mathcal{T} \mid o \in \bar{I}(t)\} =$ $D_{\bar{I}}(o) = \cup_{t \in D_I(o)} (\{t\} \cup B^+(t))$
<i>Description of a set of objects A wrt I</i>	$D_I(A)$	$D_I(A) = \cup_{o \in A} D_I(o)$

Table 2.1: Basic notions and notations

do not show the top element, it can be inferred.

2.1.2 Zoom-in

Now we will introduce elements allowing the refinement of a focus. To this end we introduce the notion of *zoom-in points*. A zoom-in point is actually a term that indicates where the user could zoom in. When building a GUI, an area is usually dedicated to each facet and the zoom-in points with respect to a facet F_i are actually those terms of \mathcal{T}_i that should be shown in that area.

Given a focus ctx , we can define its *projection* to a facet F_i , denoted by ctx_i , as follows $ctx_i = ctx \cap \mathcal{T}_i$. Now we will define the (immediate) zoom-in points with respect to a particular facet F_i . Consider a focus ctx and suppose that $ctx_i \neq \emptyset$. The *candidate*

zoom-in points with respect to F_i , denoted by $CZ_i(ctx)$, are defined as:

$$CZ_i(ctx) = N(ctx_i)$$

The above definition can also be applied in cases where $|ctx_i| > 1$, assuming that N is defined also for a set of terms S . Specifically, if $S \subseteq \mathcal{T}$ then we can define $N(S) = \cup_{t \in S} N(t)$.

From the candidate zoom-in points we now filter out those that yield an empty content. The (good or useful) *zoom-in points* are defined as:

$$Z_i(ctx) = \{t \in CZ_i(ctx) \mid \bar{I}(ctx) \cap \bar{I}(t) \neq \emptyset\}.$$

So $Z_i(ctx)$ comprises the terms of \mathcal{T}_i that should be shown in the GUI area dedicated to facet F_i if the user focus is ctx . For example, assuming the example of Figure 2.1, we have:

$$\begin{aligned} Z_1(\{Greece_1, Italy_2\}) &= \{Crete_1\} \\ Z_2(\{Greece_1, Italy_2\}) &= \{Milan_2\} \\ Z_3(\{Greece_1, Italy_2\}) &= \{6/11/2008\} \\ Z_4(\{Greece_1, Italy_2\}) &= \{[30, 150]\} \\ Z_1(\{Italy_1, Crete_2\}) &= \{Milan_1\} \\ Z_2(\{Italy_1, Crete_2\}) &= \{Iraklio_2, Chania_2\} \end{aligned}$$

When the user selects a zoom-in point t , then the current focus is updated, i.e. $ctx' = ctx \cup \{t\}$ (specifically, $ctx' = \text{minimal}_{\preceq}(ctx \cup \{t\})$). Subsequently, all new zoom-in points are computed and presented.

2.1.3 Zoom-out

The user can also *zoom out* by deselecting any term t of the corresponding focus. In that case t is replaced by its direct broader term(s) i.e. by $B(t)$. In general the replacement of t by any $t' \in B^+(t)$, or even the removal of t (without any replacement), can be considered as a zoom-out operation¹.

¹Note that if the taxonomy is a DAG then the replacement of a term t by two or more terms such that all of them subsume t , is also a zoom-out operation.

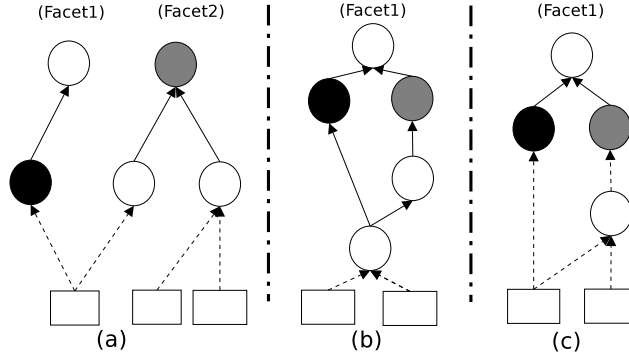


Figure 2.2: Examples of side zoom-in conditions (a) faceted taxonomies, (b) a non-tree taxonomy (i.e. DAG) and (c) multiple classification. With black are the current zoom-in points and with grey the side ones

2.1.4 Zoom-Side

Now we introduce another kind of zoom-in points. This kind of points is useful for taxonomy-based sources that satisfy at least one of the following conditions:

- (a) comprise more than one taxonomy (i.e. they are faceted taxonomies),
- (b) comprise a taxonomy that is not a tree (e.g. it is a DAG),
- (c) multiple classification (i.e. an object can be indexed with more than one terms from the same facet) is allowed with respect to at least one facet.

Figure 2.2 highlights the corresponding conditions. In all cases we assume that if user selects the black term then the grey term will be a zoom-side point. As we can see, in all conditions the black and the grey terms are incomparable with respect to the \leq and the intersection of their extension is not empty, so their union can belong to a valid *ctx*.

Definition 1 From a materialized faceted taxonomy $M = (\mathcal{F}, I)$ we can define a symmetric binary relation \rightleftharpoons over \mathcal{T} (i.e. $\rightleftharpoons \subseteq \mathcal{T}^2$), called *extensionally related*, as follows

$$t \rightleftharpoons t' \text{ iff } \bar{I}(t) \cap \bar{I}(t') \neq \emptyset \text{ and } t \parallel t'$$

where $t \parallel t'$ means that t and t' are *incomparable* with respect to \leq (i.e. neither $t \leq t'$ nor $t' \leq t$).

We can now define the *zoom-side points* w.r.t. a facet F_i , denoted by $RZ_i(ctx)$, as follows:

$$RZ_i(ctx) = \text{maximal}_{<}(\{t \in \mathcal{T}_i \mid \bar{I}(t) \cap \bar{I}(ctx) \neq \emptyset \text{ and } t \parallel ctx_i\})$$

In our running example of Figure 2.1 we have $RZ_4(\{Milan_1, Iraklio_2, [30, 90]_4\}) = \{[40, 70]_4\}$. Note that if objects are indexed by at most one term from a facet F_i and F_i is a tree, then $RZ_i(ctx) = \emptyset$ for any ctx (e.g. in our running example we have $RZ_1(\{Milan_1\}) = \emptyset$ and $RZ_2(\{Iraklio_2\}) = \emptyset$).

2.1.5 Presentation and Ranking of Zoom-in points

Each zoom-in point t is usually accompanied by a number that indicates the number of objects that will be obtained if the user selects that zoom-in point. Specifically that number equals the cardinality of the set $\bar{I}(ctx) \cap \bar{I}(t) = \bar{I}(ctx \cup \{t\})$, which is certainly greater than zero (if $t \in Z_i(ctx)$ or $t \in RZ_i(ctx)$).

The zoom-in points can be ranked according to various criteria like, number of results if selected, user preferences, popularity, usage workload, etc. Such ranking can be exploited for determining the order by which the zoom-in points are displayed in the screen, or even filtered out. Complimentarily, other criteria can also be employed to suppress the visibility of some points. For instance, we may hide those zoom-in points leading to contexts with content size below a predefined threshold, or we may decide to present only the top- K zoom-in points for each facet.

2.1.6 Restriction of a Materialized Faceted Taxonomy

As faceted exploration can be combined easily with other access methods (e.g. information retrieval queries, structured queries, or application-specific queries), the user could start interacting not only by selecting some terms (i.e. by specifying a focus), but through a set of objects, e.g. the objects returned by a full text query. To this end in this section we introduce a notion useful for capturing such scenarios.

Let $M = (\mathcal{F}, I)$ be a materialized faceted taxonomy. Let A be a subset of Obj ($A \subseteq Obj$) which could be the result of an arbitrary access method. Below we will define the *restriction of M on A* , hereafter denoted by $(\mathcal{F}, I)_{|A}$.

The *restriction of M on A* , i.e. $(\mathcal{F}, I)_{|A}$, is again a materialized faceted taxonomy, and let us write $(\mathcal{F}, I)_{|A} = (\mathcal{F}', I')$. It comprises a restriction of the interpretation function I

and a restriction of the faceted taxonomy \mathcal{F} . The later is the *reduced taxonomy*.

The interpretation I' is an interpretation that is smaller than I , denoted by $I' \sqsubseteq I$, meaning the $I'(t) \subseteq I(t)$ for each $t \in \mathcal{T}$. In particular, I' is defined as follows:

$$\forall t \in \mathcal{T}, \quad I'(t) = I(t) \cap A$$

So the range of I' is the powerset of A (and not the powerset of Obj as it is for I).

It is not hard to see that from a given interpretation I , we can define a descriptive function, denoted by $D_I(o)$ as follows:

$$\forall o \in Obj, \quad D_I(o) = \{ t \in \mathcal{T} \mid o \in I(t) \}$$

and the vice versa (i.e. from a descriptive function D we can define an interpretation I). The domain of the function D_I is the set Obj . We can restrict the domain of D_I on A , i.e. we can define the function $D_{I|A}$ (where $D_{I|A}$ denotes the restriction of the domain of D_I on A). It is equivalent to say that the interpretation I' of the restriction of M on A , is the interpretation obtained by the descriptive function $D_{I|A}$.

Now the *reduced taxonomy* \mathcal{F}' comprises a terminology \mathcal{T}' ($\mathcal{T}' \subseteq \mathcal{T}$) and a subsumption \leq' defined as follows

$$\mathcal{T}' = \{ t \in \mathcal{T} \mid \bar{I}(t) \cap A \neq \emptyset \}$$

and $\leq' = \leq \mid \mathcal{T}'$.

Equivalently,

$$\mathcal{T}' = \cup_{o \in A} B^+(D_I(o))$$

i.e. it comprises those terms that are associated with the objects in A plus all broader terms of these terms. We could denote this terminology by $\mathcal{T}_{I,A}$.

Definition 2 The *restriction of a materialized faceted taxonomy* $M = (\mathcal{F}, I)$ over a set of objects A , denoted by $(\mathcal{F}, I)_{|A}$, is again a materialized faceted taxonomy, comprising a reduced taxonomy with terminology $\mathcal{T}' = \{ t \in \mathcal{T} \mid \bar{I}(t) \cap A \neq \emptyset \}$ and an interpretation I' such that $I'(t) = I(t) \cap A$ for each $t \in \mathcal{T}$.

For example, let $M = (\mathcal{F}, I)$ be the materialized faceted taxonomy of Figure 2.1. If $A = \{Flight1, Flight2\}$ then the restriction of M on A , i.e. $(\mathcal{F}, I)_{|\{Flight1, Flight2\}}$ is shown in Figure 2.3.

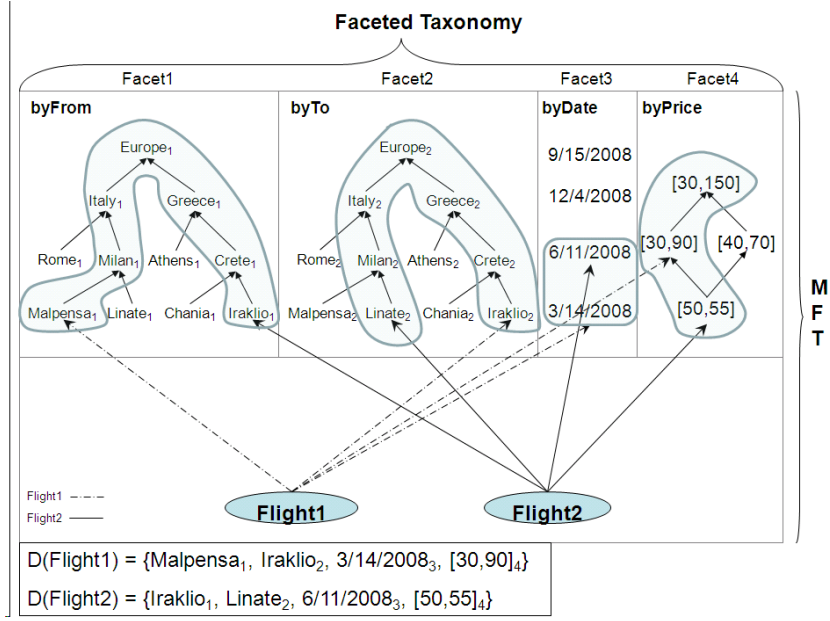


Figure 2.3: Example of a Restricted Materialized Faceted Taxonomy

2.1.7 Synopsis

Table 2.2 (taken from [39]) synopsizes and provides equivalent definitions.

2.2 Related Approaches

There are other very closely related approaches to the faceted exploration services and dynamic taxonomies which are also discussed in [40]. In this Section we report, describe and compare some of these approaches.

Formal Concept Analysis (FCA) [16] and dynamic taxonomies are closely related. As DTs are based on dynamic computations, they can tackle better dynamic collections where objects are added and deleted. In contrast, the FCA concept lattice is a static, precomputed structure that cannot easily accommodate variations in the database (apart from for incremental object insertion [31]). So FCA techniques seem more suitable for data analysis of static collections rather than for dynamic collections.

OLAP (On-Line Analytical Processing) techniques are mainly useful in cases of numerically valued facets [7]. For instance OLAP techniques like cubes could save time

Name	Notation	Definition
<i>focus</i>	ctx	any subset of T such that $ctx = \text{minimal}(ctx)$
<i>focus projection on a facet i</i>	ctx_i	$ctx_i = ctx \cap T_i$
Kinds of zoom points w.r.t. a facet i while being at ctx	Notation	Definition(s)
<i>zoom points</i>	$AZ_i(ctx)$	$= \{ t \in T_i \mid I(ctx) \cap I(t) \neq \emptyset \}$
<i>zoom-in points</i>	$Z_i^+(ctx)$	$= AZ_i(ctx) \cap N^+(ctx_i)$
<i>immediate zoom-in points</i>	$Z_i(ctx)$	$= \text{maximal}(Z_i^+(ctx))$ $= AZ_i(ctx) \cap N(ctx_i)$
<i>zoom-side points</i>	$ZR_i^+(ctx)$	$= AZ_i(ctx) \setminus (\{ctx_i\} \cup N^+(ctx_i) \cup B^+(ctx_i))$
<i>immediate zoom-side points</i>	$ZR_i(ctx)$	$= \text{maximal}(ZR_i^+(ctx))$
Restriction over an object set	Notation	Definition(s)
<i>restricted object set</i>	A	any subset of Obj
<i>reduced interpretation</i>	I'	$I'(t) = I(t) \cap A$
<i>reduced terminology</i>	T'	$= \{ t \in T \mid I'(t) \neq \emptyset \}$ $= \{ t \in T \mid \bar{I}(t) \cap A \neq \emptyset \}$ $= \cup_{o \in A} B^+(D_I(o))$

Table 2.2: Interaction notions and notations

however their construction costs time and thus are not very appropriate for dynamic collections as one would have to recompute them (although incremental maintenance is possible in certain cases). Certainly the adoption of cubes is not appropriate for facets that are computed during query answering e.g. for the facet that is derived by applying content-based clustering on the top- L document of the query answer (as in the case of **Mitos**). For the same reason, such techniques cannot be applied in cases of dynamically (e.g. user-specified) facets. Finally, OLAP techniques are not very flexible if objects are indexed with more than one terms from a facet (which is however a typical requirement of faceted classification and search). [5] discusses the relation of faceted exploration services with OLAP.

Chapter 3

Architectures and Related Work

*” Αμιλλα είναι η τάση να φτάσει κανένας τον άλλο που τον θαυμάζει
ή και να τον ξεπαράσει, χωρίς να αισθάνεται φθόνο αν ο άλλος τον ξεπερνάει.”*
Αριστοτέλης (384 p.Q. - 322 p.Q.)

This chapter is organized as follows. Section 3.1 presents the FDT interaction scheme by describing all possible states and the transitions between them. Section 3.2 elaborates on the storage policies that one can follow and discusses various algorithmic and implementation approaches regarding the realization of the exploration services. Section 3.3 presents a number of possible general architectures regarding how main and secondary memory is used. Furthermore, it discusses special index structures which have been proposed. Section 3.4 presents several web-based faceted exploration implementations and concentrates on user interfaces. Finally, Section 3.5 lists a number of faceted metadata search engines and presents an open XML specification for defining and sharing faceted classification schemes that has been proposed.

3.1 FDT Interaction & Computational Requirements

This section describes the FDT interaction scheme by presenting the possible states between client and server communication and the transitions between them. Moreover, it presents the two basic visualization modes for depicting a materialized faceted taxonomy. Finally, it discusses the main approaches for computing the zoom-in/out/side points.

3.1.1 A State-based Interaction Method

FDT interaction paradigm can be described as a state-based interaction scheme between a client (e.g. user) who wants to search and browse, and a server providing these services. Below we describe all possible states of this interaction scheme and the transitions between them.

3.1.1.1 States

Let ST denote the set of all states. A state $st \in ST$ is described by a pair (A, ctx) where A is a set of objects i.e. $A \subseteq Obj$, and ctx is a set of terms i.e. $ctx \subseteq \mathcal{T}$, $ctx = \text{minimal}_{\preceq}(ctx)$ which should satisfy the following constraints: $\emptyset \subset A \subseteq \bar{I}(ctx) \subseteq Obj$. If the user reaches a state restricted by a set of terms ctx (and not by any other external access method is used), then it will hold $A = \bar{I}(ctx)$. On the other hand, if the user reaches a state restricted by a set of objects A e.g. A has been provided by an external access method like an answer to a query in a WSE, and after that the user further restricts his scope by clicking on a set of terms ctx , then it will hold $A \subseteq \bar{I}(ctx)$.

In the sequel, we will refer to the A as $st.A$ and to the ctx as $st.ctx$.

3.1.1.2 State Visualization

While the user is at a state $st = (A, ctx)$, he is given a visualization of the restricted materialized faceted taxonomy on A , i.e. a visualization of $(F, I)|_A$. There are two visualization modes: SVM and EVM . The basic difference between the two modes is the set of terms that will be computed and presented in the UI during the interaction. In more detail:

- SVM (Simple Visualization Mode)

For each facet i the set of terms of the restriction of \mathcal{T}_i on A which will be computed and shown are defined as: $V_i(st) = B^*(st.ctx_i) \cup Z_i(st.ctx) \cup RZ_i(st.ctx)$, where the first set is the broader terms, the second the immediate zoom-in points, and the third the immediate zoom-side points.

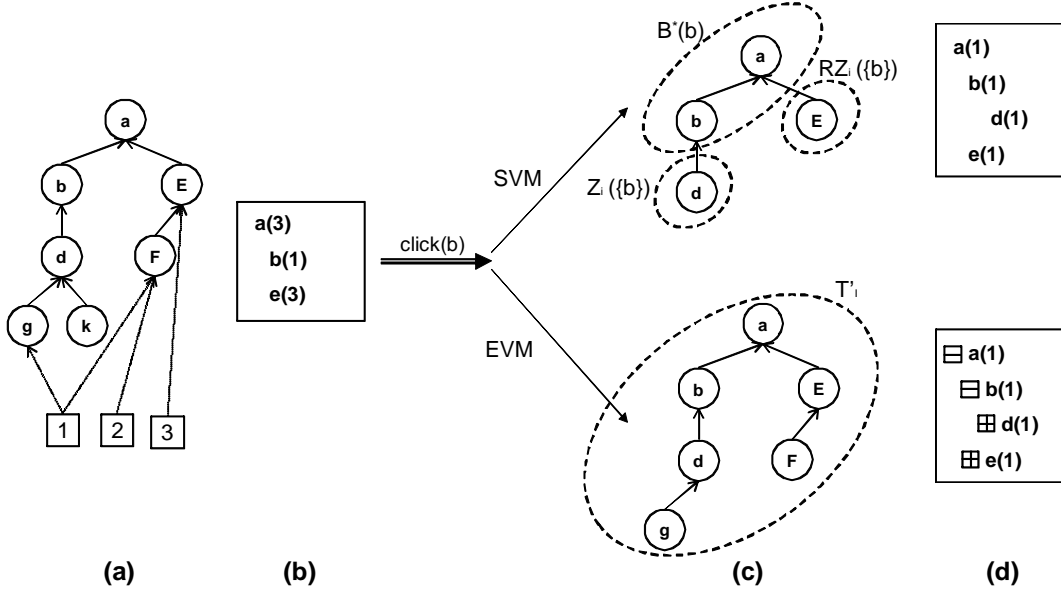


Figure 3.1: Visualization Modes Example

- *EVM* (Extended Visualization Mode)

For each facet i the set of terms which will be computed are: $EV_i(st) = \mathcal{T}'_i$ where \mathcal{T}'_i is the restriction of \mathcal{T}_i on A (remember section 2.1.6). During the interaction, *EVM* will show the $V_i(st)$, while it allows the expansion of each zoom-in point by showing the narrower zoom-in points i.e. $EV_i(st)$.

Figure 3.1(a) depicts a facet from a materialized taxonomy and how the objects of the collection are classified under facet's terms. Furthermore, Figure 3.1(b) shows the browsing structure that have been provided at the GUI layer at the current state without taking into consideration the visualization mode. Let us now assume that the user clicks on b and we want to compute the zoom-in and side points. Figure 3.1(c) sketches the terms that will be computed in each visualization mode, while Figure 3.1(d) shows the browsing structures.

3.1.1.3 State Transition

The user can change states using two operations: (i) $feed(A)$, where $\emptyset \subset A \subseteq Obj$ and (ii) $click(t)$, where $t \in V(st)$ or $t \in EV(st)$ depending on the visualization mode where $V(st) = \bigcup_{i=1}^k V_i(st)$, $EV(st) = \bigcup_{i=1}^k EV_i(st)$ and k is the number of the facets of the

materialized faceted taxonomy. Let $st' = next(st, op)$ denotes the next state if the user is on the state st and will execute the operation op . This is defined as follows:

- *Operation click(t)*

It describes the zoom operation. When the user selects/presses a zoom point t , then an operation $click(t)$ will be executed. Specifically, if $st = (A, ctx)$ and $op = click(t)$, then $st' = next(st, click(t)) = (A', ctx')$ such that $ctx' = minimal_{\preceq}(ctx \cup \{t\})$ and $A' = A \cap \bar{I}(ctx')$.

- *Operation feed(A)*

As we have already described in section 2.1.6, we can restrict a materialized faceted taxonomy M on a subset $A \subseteq Obj$ and produce the $(\mathcal{F}, I)|_A$. A user can restrict the M on A by executing the $feed(A)$ operation. In more detail, if $st = (A, ctx)$ and $op = feed(A')$, then $st' = next(st, feed(A')) = (A', ctx')$ such that $ctx' = \bigcup_{i=1}^k \top_i$.

3.1.2 General Evaluation Approaches

Independently to the visualization modes, two are the main approaches for computing the zoom-in/out/side points:

- *Extension Intersection-based* approach

In this case, the computations are based on the extension of the terms, and

- *Description-based* approach

Here, the computations are based on the descriptions of the objects which belong to A .

In order to understand the difference between the two approaches, we will show how the EV_i for a facet i can be computed using each one computation approach separately.

The EV_i with respect to the *Extension Intersection-based* approach will be all the terms $t \in \mathcal{T}_i$ such that there are objects which belong to A and are also classified under t i.e $EV_i(st) = \{t \in \mathcal{T}_i \mid st.A \cap \bar{I}(t) \neq \emptyset\}$. On the other hand, in case of *Description-based* approach they will be all the terms $t \in \mathcal{T}_i$ such that they describe an object $o \in A$ i.e. $EV_i(st) = \{t \in \bar{D}(o) \mid o \in st.A\}$.

3.2 Data Structures & Algorithms

In this section we elaborate on the storage policies that one can follow. Furthermore, we present various algorithms for computing the zoom-in/out/side points taking into account the storage policies, the evaluation approaches and the visualization methods. Moreover, we sketch their complexity and we compare them.

3.2.1 Notations

Below we assume a materialized faceted taxonomy M .

- C_M : the average number of terms that are (directly) assigned to an object $o \in Obj$, i.e. $C_M = avg_{o \in Obj}(|D(o)|)$. For instance, if we have one taxonomy and mandatory single classification then $C_M = 1$. If we have k facets and mandatory single classification with respect to each one of them, then $C_M = k$.
- $d(t)$: the depth of a term $t \in \mathcal{T}$. According to the section 2.1.1, each facet i has a top element \top_i for keeping the heads of facet's hierarchy. So, if the facet is tree-structured, then $d(t)$ is the length of the path from t to the \top_i . In case of DAG, $d(t)$ is the length of the longest path that starts from t and ends to the \top_i .
- d_{avg} : the average depth of terms in the faceted taxonomy, i.e. $d_{avg} = avg_{t \in \mathcal{T}}(d(t))$.
- $d_{M,avg}$: the average depth of terms that are directly used in object descriptions in M , i.e. $d_{M,avg} = avg_{t \in D(Obj)}(d(t))$. Notice that here we do not take into account how many objects are associated with each term. If we would like to also take that into account then, we could define the *cumulative average depth* of terms that are directly used in object descriptions:

$$d_{M,cavg} = \frac{\sum_{o \in Obj} \sum_{t \in D(o)} d(t)}{\sum_{o \in Obj} \sum_{t \in D(o)} 1}$$

and we could also refine $d_{M,cavg,i}$ analogously.

- B_{avg} : the average number of direct children that a term has. Analogously $B_{avg,i}$.
- P_{avg} : the average number of parents that a term has. Analogously $P_{avg,i}$.

- I_{avg} : the average number of objects which are directly described by a term i.e. $I_{avg} = \frac{|Obj| * C_M}{|\mathcal{T}|}$.
- \bar{I}_{avg} : the average number of objects which are described by a term i.e. $\bar{I}_{avg} = \frac{|Obj| * C_M * d_{M,avg}}{|\mathcal{T}|}$.

3.2.2 Storage Policies

In this section we describe the basic approaches one can follow for storing a materialized faceted taxonomy.

One space minimal approach is to keep stored only the reflexive and transitive reduction of the taxonomies and only the I (or equivalently D_I). An alternative, at the other extreme, approach is to store redundant (inferred) data for speeding up some computations (e.g. as it is done in [5, 38]). Specifically, we could keep stored the entire \leq of the taxonomies involved (i.e. all transitively induced relationships). Furthermore, we could keep stored the $\bar{I}(t)$ for every $t \in \mathcal{T}$ (or equivalently the $D_{\bar{I}}$ for each $o \in Obj$). In general, we should note that policies which store inferred data apart from being more memory consuming (and thus less scalable), are more expensive to maintain if changes occur.

Let's quantify the space overhead of such policies.

- overhead of \bar{I} wrt I

Let $|I|$ denote the space required for storing I i.e. $|I| = \sum_{t \in \mathcal{T}} |I(t)|$. Moreover, we know that an object $o \in Obj$ is classified under C_M terms and we assume that each object is mandatorily classified under at least one term from each facet. So, it is obvious that $|I| = |Obj| * C_M$.

If we have stored the set $\bar{I}(t), \forall t \in \mathcal{T}$, each object $o \in I(t)$ should also be stored to the extension of all $t' \in B^+(t)$. In other words, each object $o \in Obj$ will be additionally classified under $d_{M,avg}$ terms. So, it is clear that $|\bar{I}| = d_{M,avg} * |I|$, where $|\bar{I}| = \sum_{t \in \mathcal{T}} |\bar{I}(t)|$. For example, let us assume that we have the materialized faceted taxonomy presented in Figure 3.2. The materialized faceted taxonomy contains one facet which describes three objects ($|Obj| = 3$). In this case, $|I| = 3$, $|\bar{I}| = 7$, $d_{M,avg} = \frac{7}{3}$ and $C_M = 1$. So, $|\bar{I}|$ is $d_{M,avg}$ times larger than $|I|$.

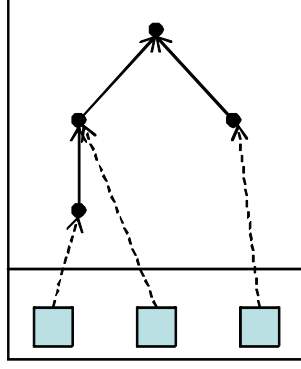


Figure 3.2: Extensions Comparison: Simple example

Here we have to mention that we do not store the $\bar{I}(\top_i)$ for a facet i .

Notice that the overhead of storing $D_{\bar{I}}$ instead of D_I , is exactly the same.

- overhead of \leq wrt \leq^r

As we have already defined, \leq^r denotes the reflexive and transitive reduction of \leq .

The number of relationships of \leq^r is at most $\frac{|\mathcal{T}|^2}{4}$. This value is obtained when

(\mathcal{T}, \leq^r) is a bipartite graph, whose $\frac{|\mathcal{T}|}{2}$ nodes are connected with all other $\frac{|\mathcal{T}|}{2}$ nodes.

On the other hand the number of relationships of \leq is at most $\frac{1}{2}|\mathcal{T}|(|\mathcal{T}| - 1)$ [45].

Alternatively, if d_{avg} is the average depth of terms in (\mathcal{T}, \leq) , then storing \leq requires storing $d_{avg} - 1$ times more relationships than \leq^r .

We will define the storage policy as a pair (X, Y) where X is \leq or \leq^r and Y is I or \bar{I} or D or \bar{D} . In general, we can say that the storage policies which can be followed are all possible combinations between X, Y . However, in case that we have stored the $\bar{I}(t)$ or $\bar{D}(t)$ where $t \in \mathcal{T}$, there is no reason for storing the \leq , as we do not need to compute and scan $N^+(t)$. On the other hand, in case we have stored the I or D and the \leq^r , the cost of computing the zoom-in points will be high as we additionally need to compute the narrower terms of t . So, in this thesis we elaborate on the below storage policies taking into consideration the evaluation approaches presented in section 3.1.2:

- *Extension Intersection-Based*

Minimal Storage Policy: (I, \leq)

Maximal Storage Policy: (\bar{I}, \leq^r)

- *Description-Based*

Minimal Storage Policy: (D, \leq)

Maximal Storage Policy: (\bar{D}, \leq^r)

3.2.2.1 Data Structures

We can store the extension of a term t ($I(t)$ or $\bar{I}(t)$), or the A in hash-based indices. Subsequently, the cost for checking whether an object $o \in A$ belongs to $\bar{I}(t)$ i.e. $o \in \bar{I}(t)$ (or the opposite), will be the cost of a lookup operation, so it takes constant time. Consequently, if we have to compute the $\bar{I}(t) \cap A$, the cost will be $\min(|A|, \bar{I}_{avg})$, while in case of $I(t) \cap A$, it will be $\min(|A|, I_{avg})$. Furthermore, in case of union e.g. $\bar{I}(t) \cup A$, the complexities will be the same as we do not want the union contains duplicate values. The algorithm for computing the union will be: we first add to the union u the maximum in size set mx ($mx = \bar{I}(t)$ or $mx = A$, taking into account their cardinalities). Let mn denotes the minimum in size set. Then for each object $o \in mn$ we check whether $o \in mx$, if no then we add it to u . This operation is the same for computing the $I(t) \cup A$.

As regards the storage of the \leq or \leq^r , we can store the relationships between the terms as sets. In more detail, if we have decided to store the \leq^r then $\forall t \in \mathcal{T}$ we will store the sets $N(t)$ and $B(t)$. On the other hand, in case of \leq , $\forall t \in \mathcal{T}$ we will store the sets $B^+(t)$ and $N^+(t)$. Finally, we will denote the cost of the union of two sets of terms as T_{\cup} while the cost of intersection as T_{\cap} .

3.2.3 Algorithms and Complexity

Here we discuss various algorithmic and implementation approaches regarding the realization of the exploration services, and we sketch and compare their complexities. The objective of this analysis is to identify good (efficient) evaluation plans taking into account the visualization modes, the general evaluation approaches and the storage policies.

3.2.3.1 EV_i & V_i Computation

Let st denotes the current interaction state. Below we present algorithms for the EV_i & V_i computation. In case of EVM we need to compute the $EV_i(st) = \mathcal{T}'$ for a facet

i. On the other hand, in *SVM*, $V_i(st) = B^*(st.ctx_i) \cup Z_i(st.ctx) \cup RZ_i(st.ctx)$, so we need to compute the immediate zoom-in and side points, and then take the union with the broader terms. According to the definition of the zoom-side points in chapter 2: $RZ_i(st) = \text{maximal}_{\leq}(\{t \in \mathcal{T}_i \mid \bar{I}(t) \cap st.A \neq \emptyset \text{ and } t \parallel st.ctx_i\})$. Let us denote the maximal incomparable terms of $t \in \mathcal{T}_i$ as $Inc_i(t)$. They will be only the brothers of t and all the brothers of its ancestors i.e. $Inc_i(t) = \{t' \in N(B^*(t))\} / \{t\}$ ¹. As the number of the $Inc_i(t)$ is dependent on the structure of the hierarchy we denote the $|Inc_i(st.ctx_i)|$ as a constant c .

- *EVM Approach*

- *Extension Intersection-based Approach*

- * *Minimal Storage Policy* (I, \leq)

To compute the EV_i for a facet i i.e. $EV_i(st)$ we have $\forall t \in \mathcal{T}_i$ to compute the $I'(t) = I(t) \cap st.A$. If $I'(t) \neq \emptyset$ then we add the set of terms $B^*(t)$ to the $EV_i(st)$ i.e. $EV_i(st) = \{t' \in B^*(t) \mid I(t) \cap st.A \neq \emptyset, t \in \mathcal{T}_i\}$. The overall cost of this approach will be the cost of taking $|\mathcal{T}|$ intersections with the $st.A$, then (in the worst case) to compute $\forall t \in \mathcal{T}$ the set $B^*(t)$ and finally to take the union of all $B^*(t)$. As we have already described, the cost of taking an intersection of I with $st.A$ will be $\min(|A|, I_{avg})$ while the cost of computing the set $B^*(t)$ for a term $t \in \mathcal{T}$ will be constant as we have stored $\forall t \in \mathcal{T}$ the $B^+(t)$. So, the overall complexity will be $|\mathcal{T}| * \min(|A|, I_{avg}) + (|\mathcal{T}| - 1) * T_{\cup}$.

- * *Maximal Storage Policy* (\bar{I}, \leq^r)

In this case, $\forall t \in \mathcal{T}$, we have to compute the $\bar{I}'(t) = \bar{I}(t) \cap st.A$. If $\bar{I}'(t) \neq \emptyset$ then $t \in EV_i(st)$ i.e. $EV_i(st) = \{t \in \mathcal{T}_i \mid \bar{I}(t) \cap st.A \neq \emptyset\}$. So, the overall cost is $|\mathcal{T}| * \min(|A|, \bar{I}_{avg})$.

- *Description-based Approach*

- * *Minimal Storage Policy* (D, \leq)

¹In case of DAG, we use the \top element for each facet, so we can visit paths that visually they do not have any common ancestor.

Here, $\forall o \in st.A$ we have to get the terms $t \in D(o)$, and then to compute the $B^*(t)$. Finally, we have to take the union of all $B^*(t)$ i.e $EV_i(st) = \{t' \in B^*(t) \mid t \in D(o), \forall o \in st.A\}$. The cost of getting the terms $t \in D(o)$ will be $|A| * \frac{C_M}{k}$, where k is the number of facets, as each object will be directly classified under $\frac{C_M}{k}$ terms from each facet. The cost of computing the $B^*(t)$ is constant while the cost of computing the union will be $(|A| * \frac{C_M}{k} - 1) * T_{\cup}$. So, the overall complexity will be $|A| * \frac{C_M}{k} + (|A| * \frac{C_M}{k} - 1) * T_{\cup}$.

* *Maximal Storage Policy* (\bar{D}, \leq^r)

In this case the zoom-in points will be the terms $t \in \bar{D}(st.A)$. Formally, $EV_i(st) = \{t \in \bar{D}(o), \forall o \in st.A\}$. Also in this case we have to take the unions of all $\bar{D}(o)$. The overall cost will be the same as in minimal storage policy, so $|A| * \frac{C_M}{k} + (|A| * \frac{C_M}{k} - 1) * T_{\cup}$.

- *SVM Approach*

- *Extension Intersection-based Approach*

* *Minimal Storage Policy* (I, \leq)

In this case, we need to compute $\forall t \in N(st.ctx_i) \cup Inc_i(st.ctx_i)$ the $\bar{I}'(t) = \bar{I}(t) \cap st.A$. If $\bar{I}'(t) \neq \emptyset$ then $t \in V_i(st)$. Finally we need to add to $V_i(st)$ the $B^*(st.ctx_i)$ i.e $V_i(st) = \{t \in N(st.ctx_i) \cup Inc_i(st.ctx_i) \mid \bar{I}(t) \cap st.A \neq \emptyset\} \cup B^*(st.ctx_i)$. The complexity of this approach will be the cost of computing the $Inc_i(st.ctx_i)$ and the $\bar{I}(t)$ from $I(t)$ and then take $|N(st.ctx_i) \cup Inc_i(st.ctx_i)|$ intersections. Moreover, we need to compute the $B^*(st.ctx_i)$ and add it to the V_i . It is easy to see that $|N(st.ctx_i)| = B_{avg}$, while the cost for computing the $Inc_i(st.ctx_i)$ or the $B^*(st.ctx_i)$ will be constant as we have stored the \leq^2 . The cost of intersection is $\min(|A|, \bar{I}_{avg})$. The cost of computing the $\bar{I}(t)$ will be $(d_{avg} - 1) * I_{avg}$, as we have to compute the $\bigcup_{t' \in N^*(t)} I(t')$. Consequently, the overall complexity will be $(B_{avg} + c) * (\min(|A|, \bar{I}_{avg}) + (d_{avg} - 1) * I_{avg})$.

* *Maximal Storage Policy* (\bar{I}, \leq^r)

²We do not need transitive closure computations.

Here, $\forall t \in N(st.ctx_i) \cup Inc_i(st.ctx_i)$ we have to compute the intersection $\bar{I}(t) \cap st.A$ and then add to V_i the $B^*(st.ctx_i)$ i.e. $V_i(st) = \{t \in N(st.ctx_i) \cup Inc_i(st.ctx_i) \mid \bar{I}(t) \cap st.A \neq \emptyset\} \cup B^*(st.ctx_i)$. As we have stored the \leq^r the cost of computing the $B^*(st.ctx_i)$ and $Inc_i(st.ctx_i)$ will not be constant. In more detail, the cost of computing the $B^*(st.ctx_i)$ will be $P_{avg} * T_{\cup}$ as we need to compute the union of all $B(p)$ where $p \in P_{avg}$. The cost of $Inc_i(st.ctx_i)$ computation will be the cost to compute the $N(t)$, $\forall t \in B^*(st.ctx_i)$. As we have already stored the $N(t)$ $\forall t \in \mathcal{T}$ we will not have any additional cost. So in this case the complexity is $(B_{avg} + c) * \min(|A|, \bar{I}_{avg}) + P_{avg} * T_{\cup}$.

– *Description-based Approach*

* *Minimal Storage Policy* (D, \leq)

Here, we will use the same technique as in the *EVM* approach (*EVM* approach, *Description-based, Minimal Storage Policy*). The only difference is that we need to take the intersection with the $N(ctx_i) \cup Inc_i(st.ctx_i) \cup B^*(st.ctx_i)$ i.e. $V_i(st) = \{t' \in B^*(t) \mid t \in D(o), \forall o \in st.A\} \cap (N(st.ctx_i) \cup Inc_i(st.ctx_i) \cup B^*(st.ctx_i))$. So the complexity is $|A| * \frac{C_M}{k} + (|A| * \frac{C_M}{k} - 1) * T_{\cup} + T_{\cap}$.

* *Maximal Storage Policy* (\bar{D}, \leq^r)

Also, in this case we follow the corresponding *EVM* approach (*EVM* approach, *Description-based, Maximal Storage Policy*) and we take the intersection with $N(st.ctx_i) \cup Inc_i(st.ctx_i) \cup B^*(st.ctx_i)$ i.e. $V_i(st) = \{t \in \bar{D}(o), \forall o \in st.A\} \cap (N(st.ctx_i) \cup Inc_i(st.ctx_i) \cup B^*(st.ctx_i))$. As we have already computed, the cost of $B^*(t)$ and $Inc_i(st.ctx_i)$ computation is $2 * P_{avg} * T_{\cup}$. Subsequently, $|A| * \frac{C_M}{k} + (|A| * \frac{C_M}{k} - 1) * T_{\cup} + T_{\cap} + P_{avg} * T_{\cup}$.

3.2.3.2 Zoom-out points Computation

Basically, a zoom-out operation can be executed only after a zoom-in operation as it is exactly the opposite. In more detail, we can say that when we zoom-out we execute a *click*(t') operation where t' is broader than the term t which was clicked in the last

click operation. We can say that the computation of zoom-out points can be done in constant time as any term which is broader than a zoom-in point will be valid too. This holds because the objects which are classified under a term t , are also classified under the broader terms of t .

In contrast to the above, let us assume that we are in the state $st = (A, ctx)$ and we have computed the zoom-in or the immediate zoom-in points. Furthermore, imagine that the user has the ability to zoom-out from the current focus. For instance, if $st.A = st_{prev}.A \cap \bar{I}(st.ctx)$ where st_{prev} denotes the previous state, the new state st' with the zoom-out operation will be $st' = (A', ctx)$ where $A' = \bar{I}(st.ctx)$. In this case, the computation of the zoom-out points will have the same complexity as the computation of zoom-in operations, as the $st'.A$ has been changed.

3.2.3.3 Count Information

In case we follow the *Extension Intersection-based* approach, we can define the count of a term t as $count(t) = |\bar{I}(t) \cap A|$, while in case of *Description-based* as $count(t) = tf_{t, \bar{D}_i(A)}$ where $tf_{t, \bar{D}_i(A)}$ is the appearance frequency of t in $\bar{D}_i(A)$.

If we follow the *maximal storage* policy independently to the evaluation approach, the complexity of computing the count of a term t will be constant as we have already made these computations in order to decide if t is a zoom-in point.

In case of *minimal storage* policy, if we follow the *Extension Intersection-based* approach we need to compute the $\bar{I}(t)$ from $I(t)$ and then take the intersection. We have already pay this cost only in case of *SVM*. On the other hand, in case of *Description-based* approach, we need to compute how many times the t appears in $\bar{D}_i(A)$.

However, we can avoid these computations by providing the count information of a term approximately. Section 5.2.1.5 presents a method for providing approximately the count information of a term in constant time.

3.2.3.4 Conclusions of the Analysis

Table 3.1 presents the complexity of the zoom-in computation taking into account the storage policies, the visualization methods and the computation approaches.

If we follow the *Description-based* approach, it is obvious that we do not need to follow a *maximal storage* policy as the computational costs are almost the same as in case of *minimal storage* policy. However, the basic drawback of the *minimal storage* policy is that the cost of zoom-in points computation does not include the cost of computing the count information. To provide the exact count information for each zoom-in point we need to follow the algorithms which follow the *maximal storage* policy.

In the *Extension Intersection-based* approach, the computation of zoom-in points according to the *maximal storage* policy costs less than the case of *minimal*, and it also contains the cost of providing the count information. However, the storage overhead is bigger.

Finally, we need to specify which evaluation approach is preferable for very large collections e.g. $|Obj| = 10^{10}$. We can see that the complexity of *Description-based* evaluation approach is always proportional to $|A|$, while in *Extension Intersection-based* it is proportional to $\min(|A|, I_{avg})$ or $\min(|A|, \bar{I}_{avg})$. It is obvious that if A is very large then the *Description-based* approach is prohibitive. On the other hand, if A is small, the *Description-based* approach seems to be better as it is independent to the size and structure of the facet hierarchy. Specifically, in cases that we have mandatory single classification then $\frac{C_M}{k} = 1$.

Storage Policy	Complexity
<i>EVM</i>	
(I, \leq)	$ \mathcal{T} * \min(A , I_{avg}) + (\mathcal{T} - 1) * T_{\cup}$
(I, \leq^r)	$ \mathcal{T} * \min(A , I_{avg})$
(D, \leq)	$ A * \frac{C_M}{k} + (A * \frac{C_M}{k} - 1) * T_{\cup}$
(\bar{D}, \leq^r)	$ A * \frac{C_M}{k} + (A * \frac{C_M}{k} - 1) * T_{\cup}$
<i>SVM</i>	
(I, \leq)	$(B_{avg} + c) * (\min(A , I_{avg}) + (d_{avg} - 1) * I_{avg})$
(I, \leq^r)	$(B_{avg} + c) * \min(A , I_{avg}) + P_{avg} * T_{\cup}$
(D, \leq)	$ A * \frac{C_M}{k} + (A * \frac{C_M}{k} - 1) * T_{\cup} + T_{\cap}$
(\bar{D}, \leq^r)	$ A * \frac{C_M}{k} + (A * \frac{C_M}{k} - 1) * T_{\cup} + T_{\cap} + P_{avg} * T_{\cup}$

Table 3.1: Zoom-in points Computation' Complexities

3.3 Possible Architectures

Here we distinguish two general architectures regarding how main and secondary memory is used as we are interested in very large data sets. Each architecture has different applicability and pos and cons.

3.3.1 (MEM) Architecture

In this architecture all data are kept in Main Memory. As faceted exploration can be combined easily with other access methods (e.g. information retrieval queries, structured queries, or application-specific queries), another variation of the (MEM) architecture is possible: to load in main memory only the answer of each submitted query. Below we focus on specialized index structures which have been proposed for implementation and follow the (MEM) architecture.

The implementation described in [5] uses Apache Lucene web search engine library, and Apache Solr which is an open source enterprise search server based on Lucene that deals with non-hierarchical facets. In that approach, a taxonomy T is a DAG whose nodes represent facet terms and direct edges denote the specialization (refinement) relations between them. T is stored into a structure called *Taxonomy Index*. Let assume that we have the materialized faceted taxonomy of Figure 3.3, the *Taxonomy Index* which will be created is shown at Figure 3.4.

The interpretations of terms are stored in a inverted index, i.e for each term t , the set $\bar{I}(t)$ is kept into a posting list and t is described by its taxonomy path. Another postings list named *DirectIndex* stores the description of each object o (having $id=oid$) w.r.t. to I , i.e $D_I(o)$, being a list of term ids (tid). Figure 3.4 presents the indices that will be created for the facet "By Location".

A more efficient implementation for large information bases is presented in [38]. To store the interpretation of each term $t \in \mathcal{T}$, for each object $o \in \bar{I}(t)$ a tuple $De(tid, oid)$ is stored, where oid is the *id* of the object o and tid is the *id* of the term t . In order to store the De tuples, that work exploits the observation that a term ht at the high levels of a facet hierarchy will belong to the majority of the descriptions of the objects, while a

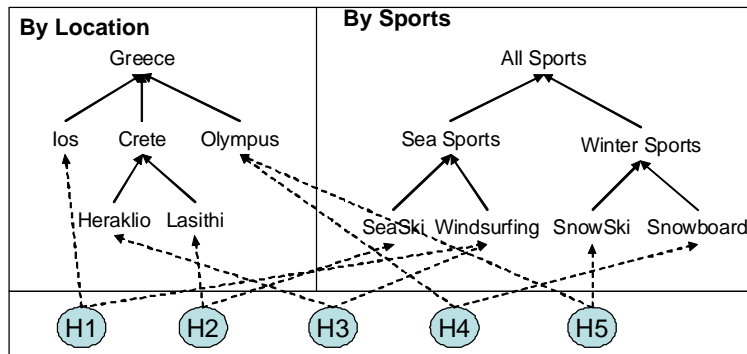


Figure 3.3: A simple MFT

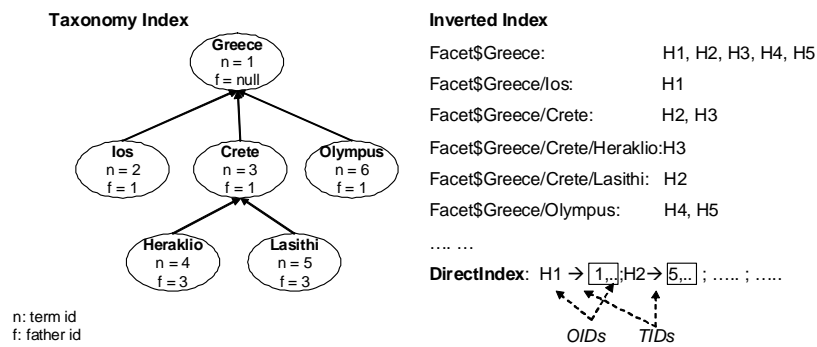


Figure 3.4: Storage indices according to [5]

term lt at low levels (e.g. leaves) will belong to few descriptions of objects. So is better to have different kind of indices for the highest and lowest levels with respect to the storage overhead.

So, the author proposes the following compression strategy. For the higher levels, he uses a bitmap of $|HT| \times |Obj|$ dimensions where HT is the set of all $ht \in \mathcal{T}$ w.r.t. \leq . If $o \in \bar{I}(t)$ he put 1 in $[tid, oid]$ cell; 0 otherwise. For the lower levels, the author uses an inverted list where for each term lt , an ordered vector with every $oid \in \bar{I}(lt)$ is kept (as in [5]). In order to have a single index for all terms, he uses a pointer array keyed by tid where if the term at the position i of the table is a highest level term the pointer points to a specific row of the bitmap, otherwise it points to a specific position of the inverted list. Figure 3.5 depicts the indices that this approach will create for the example of Figure 3.3 for the facet "ByLocation", assuming that $HT = \{Greece, Ios, Crete, Olympus\}$.

For the taxonomies of facets, [38] uses the below indices:

- a father-to-son structure, FS , which for each $t \in \mathcal{T}$ it stores $N(t)$ i.e for each tid it stores the sequence of its sons, ordered by display order.
- a son-to-father structure, SF , which $\forall t \in \mathcal{T}$ it stores the $B(t)$ i.e for each term tid it stores the set of its fathers in case the hierarchy is a DAG, or its single father if hierarchy is a tree. This structure allows upwards navigation from a term to the taxonomy top element.
- a *Descendants* structure which for each term stores the set of all its descendants, i.e $\forall t \in \mathcal{T}$ stores the $N^+(t)$. and
- an *Ancestors* structure which for each term it stores the set of all its ancestors, i.e $\forall t \in \mathcal{T}$ stores the $B^+(t)$.

3.3.2 (DB) Architecture

Here we examine the case where all data are stored in a relational database. The motivation for elaborating on this case is that relational database technology dominates in

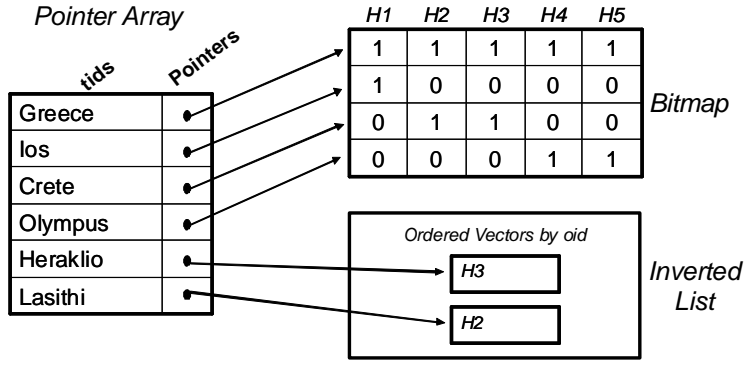


Figure 3.5: Storage indices according to [38]

business applications. However, we should mention that this approach is feasible only if we a-priori know the depth of the taxonomies involved or if we adopt recursive SQL.

An implementation which follows this approach is the Flamenco project[59]. It relies on a relational DBMS (specifically MySQL) and for each object $o \in Obj$ it stores $D(o)$ using tuples at the form (o, t) for each term $t \in D(o)$. Flamenco does not store the \bar{I} of terms and has to dynamically reconstruct it by taking the union of the I of narrower terms. When the user selects a zoom-in point, a query is generated using the *SQL COUNT(*)* and *GROUPBY* operators to count the number of objects that fall into each facet term. Further implementations which are based on RDBMS include i411³ and Atomz⁴.

At first, in section 3.3.2.1 we describe in brief some methodological issues regarding the application of the faceted exploration paradigm over relational databases. Furthermore, section 3.3.2.2 describes how SQL can be used for enabling information exploration services.

3.3.2.1 From the Relational to the Faceted Data Model: Methodological Comments

Suppose that we have a relational database and we want to offer faceted exploration services for its contents.

One approach would be to define a view containing the attributes that should be considered as facets. This means that the declarative query language offered by a DBMS

³Source: <http://www.i411.com>

⁴Source: <http://www.atomz.com>

can be exploited for defining the desired facets, i.e. those that are appropriate for browsing by humans. Note that the relational view may comprise attributes coming from different relations (and its definition may include joins and other transformations). Each object $o \in Obj$ is represented as a tuple, while each attribute of that view is considered as a facet, and the set of distinct values of these attributes that appear in the tuples of the view are considered as the terms of that facet.

Of course, the paradigm of faceted exploration can be combined with other existing methods: e.g. with predefined query forms or with plain SQL query answering. In that case, faceted exploration can be used to summarize the results of these access methods. It is not necessary for the faceted view to include all attributes that characterize an object, or all the attributes that are being exploited by other access methods. It may contain only those that are appropriate for exploration.

However, some frequently occurring attributes, like "price", "weight", "dates", "locations", usually have a big number of distinct values (which are not hierarchically organized). It would be problematic (in terms of usability) to visualize all such values as candidate zoom-in points. To alleviate this problem, an additional step that aims at organizing these values hierarchically could be adopted. Such hierarchies can be defined manually or automatically. For instance, there may already exist appropriate hierarchies which could be stored in the DB (represented as separate relational tables). Alternatively, automatic methods for defining hierarchies could be adopted. For instance, [6] describes methods for creating multi-level taxonomies for attribute values on the fly. In general, a number of techniques for creating such hierarchies for frequently occurring cases and needs, could be developed and supported. Some indicative examples are given in Table 3.2. The table also shows how the children of a node (hence the set of its zoom-in points) could be ordered (the listed choices could be considered as alternative/complementary/optional criteria to the "default ordering mode" which usually is: order values in descending order with respect to the number of hits). Apart from such (simple) cases, there is almost always the trade-off between degree of automation and quality of produced hierarchy.

Attribute	Possible hierarchies of attribute values	Ordering of children
Prices	Intervals of prices	According to their value
Dates and Periods	Years, months, dates e.g. 2008-05-21, 2008-05-22 2008 → 05 → 21 22	According to their value
Place Names	Countries, Regions, Cities, Suburbs, Streets, Interval of street numbers,	Lexicographically
Web Domains (in general strings formed according to a hierarchical naming scheme)	$GR \rightarrow FORTH \rightarrow ICS \rightarrow ISL$	Lexicographically

Table 3.2: Automatic Hierarchy Creation Examples

3.3.2.2 On SQL implementation

This section describes how SQL can be used for realizing the exploration services. This is done over a running example that includes an hierarchy that is represented and stored in the database. Consider the following schema:

`Hotel(hId, hName, stars, lId)`

`Location(lid, lName, parentlId)`

with the following foreign key constraints

`Hotel.lid ⊆ Location.lid`

`Location.parentlId ⊆ Location.lid`

and assume the domain of the attribute **stars** in the integer interval [1..5].

We can consider this database schema as a materialized faceted taxonomy $F = (F_h, F_s, F_l)$ with three facets corresponding to the attributes (hotel) **name**, **stars** and **location** respectively. We can define $\mathcal{T} = \mathcal{T}_h \cup \mathcal{T}_s \cup \mathcal{T}_l$ where \mathcal{T}_h comprise the names of the **Hotels**, \mathcal{T}_s contains those values of [1..5] that occur in the database, and \mathcal{T}_l are the location names in the relation **Location**.

Let us assume that the terms of F_l are hierarchically organized as follows: **Crete** < **Greece** < **Europe**, and **Italy** < **Europe**. For example, the table **Location** could have

the following contents:

Location		
lid	lName	parentlId
1	Europe	NULL
2	Greece	1
3	Italy	1
4	Crete	2

Below we will present queries for computing the zoom-in/out/side points according to the approaches presented in section 3.2.

3.3.2.3 Direct and Indirect Narrower/Broader terms of a term

Let us assume that our materialized faceted taxonomy contains the facet i which is not hierarchically organized. Subsequently, for every $t \in \mathcal{T}_i$ we have $N(t) = \emptyset$ and $N^+(t) = \emptyset$. On the other hand, if the facet i is a hierarchy of values then $N(t)$ can be computed with one selection query, while $N^+(t)$ can be computed with a recursive approach. For example, to compute $N(EuropeId)$ we can use the query $\Pi_{lid}(\sigma_{parentlId=EuropeId}(Location))$, i.e.:

```
SELECT lid FROM Location WHERE parentlId=EuropeId
```

and we can denote this query by $q_N^{(1)}(EuropeId)$.

The direct broader terms of a term, e.g. of **Crete**, can be computed analogously, by $\Pi_{parentlId}(\sigma_{lId=CreteId}(Location))$:

```
SELECT parentlId FROM Location WHERE lId=CreteId
```

and we can denote this query by $q_B^{(1)}(CreteId)$.

In case we have to compute the indirect narrower/broader terms of a term t e.g. $N^+(t)$ or $B^+(t)$, we need to define the number of the links between the t and its narrower/broader terms. Let denote it as d . We need this assumption as we need a-priori know the depth of the hierarchy. It is clear that in case of indirect narrower/broader terms $d > 1$, while in case of direct $d = 1$. Then $q_N^{(d)}(t)$ contains the narrower terms of t at exactly d links ($<$) distance. Let $q_N^{(1)}(t)$ denote the query template "SELECT lid FROM Location WHERE

parentlId IN t ". Then we can write $q_N^{(2)}(EuropeId) = q_N^{(1)}(q_N^{(1)}(EuropeId))$. We can generalize and construct such queries for various values of d as follows:

$$q_N^{(d)}(t) = q_N^{(1)}(q_N^{(d-1)}(t))$$

Analogously we can define the query $q_B^{(d)}(t)$.

If we want all narrower (resp. broader) terms that can be reached with **at most** d links, we just have to change the query $q_N^{(1)}(t)$ (resp. $q_B^{(1)}(t)$). Specifically, in that case $q_N^{(1*)}(t)$ should denote the template "SELECT lid FROM Location WHERE lid IN t OR parentlId IN t ".

Hereafter we can use the notations $q_N^{(d*)}(t)$ and $q_B^{(d*)}(t)$ to denote such queries.

3.3.2.4 Maximal Incomparable Terms of a Term

As we presented in section 3.2.3.1, we denote the maximal incomparable terms of a term t as $Inc_i(t) = \{t' \in N(B^*(t))\}/\{t\}$. In this section we present sql queries which compute the $Inc_i(st.ctx_i)$. The query in our running example would be:

```
SELECT lid
FROM Location
WHERE lid IN (q_N^{(1)}(q_B^{(d*)}(st.ctx_i)) MINUS st.ctx_i)
```

Here we have to remind that we need to compute the $Inc_i(st.ctx_i)$ only in case of *Simple Visualization Mode*.

3.3.2.5 Direct and Indirect Narrower/Broader terms of a set of terms

Let s a set of terms i.e. $s \subseteq \mathcal{T}$. We can write that

$$\begin{aligned} N(s) &= \cup_{t \in s} N(t) \\ B(s) &= \cup_{t \in s} B(t) \end{aligned}$$

We can extend the above queries so that to compute these sets by adding a disjunction. For example $N(\{GreeceId, ItalyId\})$ can be computed by the query:

$\Pi_{lid}(\sigma_{parentlId \in \{GreeceId, ItalyId\}}(Location))$, i.e.:

```
SELECT lid FROM Location WHERE parentlId IN {GreeceId, ItalyId}
```

and we can denote this query by $q_N^{(1)}(\{GreeceId, ItalyId\})$. Analogously we can define the query $q_B^{(1)}(s)$.

As in case of the computation of the indirect narrower/broader terms of a term t , we can define the query $q_N^{(d)}(s)$ for computing the narrower set of terms of s at exactly d links ($<$) distance. Let $q_N^{(1)}(s)$ denote the query template "SELECT lid FROM Location WHERE parentId IN s ". Then we can write $q_N^{(2)}(\{EuropeId, CreteId\}) = q_N^{(1)}(q_N^{(1)}(\{EuropeId, CreteId\}))$. We can generalize and construct such queries for various values of d as follows:

$$q_N^{(d)}(s) = q_N^{(1)}(q_N^{(d-1)}(s))$$

Analogously we can define a query $q_B^{(d)}(s)$.

In addition, if we want all narrower (resp. broader) terms that can be reached with **at most** d links, we just have to change the query $q_N^{(1)}(s)$ (resp. $q_B^{(1)}(s)$). Specifically, in that case $q_N^{(1)*}(s)$ should denote the template "SELECT lid FROM Location WHERE lid IN s OR parentId IN s ".

Hereafter we can use the notations $q_N^{(d*)}(s)$ and $q_B^{(d*)}(s)$ to denote such queries.

3.3.2.6 Model Interpretations

A context ctx is any subset of \mathcal{T} . Suppose that $ctx = \{t_1, \dots, t_k\}$ where $t_i \in \mathcal{T}_i$ and each \mathcal{T}_i is the domain of a relational attribute A_i . For computing $I(ctx)$ we can define a selection condition, denoted by ϕ_{ctx} , defined as:

$$\phi_{ctx} : (A_1 = t_1) \wedge \dots \wedge (A_k = t_k)$$

For example, if $ctx = \{Sunwing, 4\}$ then $\phi_{ctx} = \text{"hname=Sunwing AND stars = 4"}$. We can compute $I(ctx)$ using the selection query $\sigma_{\phi_{ctx}}(Hotel)$.

Analogously we can construct selection conditions for terms corresponding to location ids. However, if a term corresponds to a location name, e.g. if $ctx = \{Crete\}$, then we need a query that includes a join, specifically the query $\sigma_{lname=Crete}(Hotel \bowtie Location)$ corresponding to the SQL query:

```
SELECT * FROM Hotel, Location WHERE lname="Crete" AND
      Hotel.lid = Location.lid
```

Furthermore, we will use ϕ_{ctx_i} to denote the corresponding selection condition that concerns facet F_i only. So, to compute $I(t)$ where $t \in \mathcal{T}_s$ we can use the query $\sigma_{\phi_t}(Hotel)$. For example, if $t = 3$ the query will be:

```
SELECT *
FROM Hotels
WHERE stars=3
```

Consider now a term t that is part of a taxonomy with depth d . In this case we can write:

$$\bar{I}(t) = \cup \{ I(t') \mid t' \leq t \} = \cup \{ I(t') \mid t' \in N^{(d*)}(t) \}$$

where $N^{(d*)}(t), t \in \mathcal{T}$ contains the t and all its narrower terms at exactly d links ($<$) distance.

To compute $\bar{I}(t)$, e.g. $\bar{I}(EuropeId)$, we can use the query:

$$\pi_{hId, hName, stars, lId}(Hotels \bowtie_{Hotels.lId=Location.lId} R_x)$$

where $R_x = \sigma_{lid \in q_N^{(d*)}(EuropeID)}(Location)$

or

```
SELECT *
FROM Hotels
WHERE lid IN q_N^{(d*)}(EuropeID)
```

For more than one terms we have to use conjunction. For example to compute

$\bar{I}(\{EuropeId, 3stars\})$ we can use the query:

let $R_x = \sigma_{stars=3}(Hotels)$ and

$R_y = \sigma_{lid \in q_N^{(d*)}(EuropeID)}(Location)$ we have

$\pi_{hId, hName, stars, lId}(R_x \bowtie_{Hotels.lId=Location.lId} R_y)$ i.e.:

```
SELECT *
FROM Hotels
WHERE stars = 3 AND lid IN q_N^{(d*)}(EuropeID)
```

Here we have to mention that the order of AND operations leaves space for optimization. For example, if we firstly write the case $stars = 3$, we will execute the R_y only RX times, where RX denotes the cardinality of the result set if we execute the R_x query. On the other hand, if they have the opposite order it will be executed $F_{count(*)}(Hotels)$ times.

To sum up, if $ctx = \{t_1, \dots, t_k\}$ where $t_i \in \mathcal{T}_i$ and all \mathcal{T}_i are hierarchically organized with

maximum depth d , then $\bar{I}(ctx)$ can be computed with a query that has as condition

$$\bar{\phi}(ctx) = \bigwedge_{i=1}^k A_i \text{ IN } q_N^{(d*)}(t_i)$$

3.3.2.7 Object Descriptions

We can define the description of an object oid with respect to a facet F_i as follows: $D_i(oid) = \{t \in \mathcal{T}_i \mid o \in I(t)\}$. It can be computed by a projection on a selection query $\Pi_{A_i}(\sigma_{id=oid}(R)) \equiv q_i(o)$. For example, $D_L(h2)$ can be computed by the query $q_L(h2) = \text{"select lid from hotels where hid=h2"}$.

We can define the *complete description* of an object oid wrt a facet F_i as follows: $\bar{D}_i(oid) = B^+(D_i(o))$. For example for a hotel $h1$ located in *Crete* we have $\bar{D}_L(h1) = \{Crete, Greece, Europe\}$. If the maximum depth is d then the complete description can be computed by the query $q_B^{(d*)}(Crete)$. Specifically by the query $q_B^{(d*)}(q_i(h1))$.

3.3.2.8 Complete Descriptions

As we have already described, we can compute the $\bar{D}_i(o), o \in Obj$ or the $\bar{I}(t), t \in \mathcal{T}$ by using specific forms of queries. However, these computations can be used in the *minimal storage policy* where the D, I are stored. In case we want to follow the *maximal storage policy* we need to construct the complete descriptions of the objects. Then the queries will be simpler and faster. Below, we will present the queries for constructing the complete descriptions.

To begin with, let us assume that we have a database with the same relational schema as in our running example and we add some hotels. The tuples of the tables are shown in Figure 3.6(a). To create the complete descriptions we need to create an additional table for each domain which is hierarchically organized. In our example, we need to create a table for the location. The new table will have two domains which will be both primary keys, hid and lid . Then for each hotel, we need to add one tuple with the id of the hotel and the id of hotel's specific location and a tuple for each broader location. Figure 3.6(b) depicts the table *CompleteDescriptions* which contains the complete descriptions of the hotels for the domain *location*. For example, for the hotel *Minoan Palace* which is located in *Crete*, three tuples will be created: (4,1) for *Europe*, (4,2) for *Greece* and (4,4) for *Crete*.

We can create the complete descriptions for an object $o \in Obj$ with the following way. Let us assume that we need to compute the complete descriptions of the *Minoan Palace* hotel. First,

Location		
lid	Iname	parentId
1	Europe	Null
2	Greece	1
3	Italy	1
4	Crete	2

Hotel			
hid	hname	stars	lid
1	Relax	3	1
2	Poseidon	4	2
3	Colosseum	5	3
4	Minoan Palace	4	4

CompleteDescriptions	
hid	lid
1	1
2	1
2	2
3	1
3	3
4	1
4	2
4	4

(a)
(b)

Figure 3.6: Constructing Complete Descriptions

we need to compute all the broader locations of the hotel. We can do it by computing the $Locs = q_B^{(d^*)}(q_L(4))$. Then for each location l which belongs to the result of the query $Locs$ we can create an insert query which will have the following form:

```
INSERT INTO CompleteDescriptions(hid,lid)
VALUES (4, l)
```

Also we have the ability to construct the complete descriptions of all objects for the location domain by executing a query with the following form:

```
INSERT INTO CompleteDescriptions(hid,lid)
VALUES(hid,q_B^{(d^*)}(q_L(hid)))
```

Now, we can compute the $\bar{I}(t)$ and $\bar{D}(o)$ faster. For example the query for computing the $\bar{I}(EuropeId)$ would be:

```
SELECT hid
FROM CompleteDescriptions
WHERE lid = EuropeId
```

On the other hand, the query for computing the $\bar{D}_L(4)$ would be:

```
SELECT lid
FROM CompleteDescriptions
WHERE hid = 4
```

3.3.2.9 V_i & EV_i Computation

As in section 3.2.3.1, to compute the V_i , EV_i we have to take into account the storage policy (*minimal* or *maximal*), the visualization mode (*EVM* or *SVM*) and the evaluation approach (*Extension Intersection-based* or *Description-based*). Furthermore, in (DB) architecture we have to decide where the computations of intersection and unions will be executed. In more detail, we have two possible approaches. In the first approach, all computations are executed by SQL queries. A second approach would be to execute queries which compute the D, I, \bar{I}, \bar{D} , then we get the result sets and finally we execute the intersections and unions in main memory. In this thesis we elaborate on the first approach.

Let st denotes the current interaction state. We will not give any detail about flat facets as the queries for computing the zoom-in points are somewhat trivial (use of DISTINCT and GROUP BY operators). Below we present queries for the zoom-in points computation in hierarchically organized domains. So, we want to compute the zoom-in points for the *Location* facet. The $st.A$ has to be computed in each interaction state, as we have not stored it in main memory. In the sequel, we use the notation $st.A$ but we have to mention that it is a sub-select query.

- *EVM* Approach

- *Extension Intersection-based* Approach

- * *Minimal Storage Policy* (I, \leq)

In this case $EV_i(st) = \{t' \in B^*(t) \mid I(t) \cap st.A \neq \emptyset, t \in \mathcal{T}_i\}$. Query:

```
SELECT DISTINCT(Location.lname)
FROM Location
WHERE Location.lid IN  $q_B^{(d*)}(st.ctx_i)$  AND
(SELECT COUNT(*) FROM Hotel WHERE Hotel.lid=Location.lid
INTERSECT st.A ) > 0
```

- * *Maximal Storage Policy* (\bar{I}, \leq^r)

In this case, $EV_i(st) = \{t \in \mathcal{T}_i \mid \bar{I}(t) \cap st.A \neq \emptyset\}$. Query:

```
SELECT DISTINCT(Location.lname)
FROM Location
WHERE
```

```
(SELECT COUNT(*) FROM CompleteDescription WHERE lid = Location.lid
INTERSECT st.A ) > 0
```

– *Description-based Approach*

* *Minimal Storage Policy* (D, \leq)

Here, $EV_i(st) = \{t' \in B^*(t) \mid t \in D(o), \forall o \in st.A\}$. Query:

```
SELECT DISTINCT(Location.lname)
FROM Location
WHERE Location.lid IN  $q_B^{(d*)}$ (SELECT DISTINCT(lid) FROM Hotel
where hid IN st.A)
```

* *Maximal Storage Policy* (\bar{D}, \leq^r)

In this case, $EV_i(st) = \{t \in \bar{D}(o), \forall o \in st.A\}$. Query:

```
SELECT DISTINCT(Location.lname)
FROM Location, CompleteDescription
WHERE CompleteDescription.hid IN st.A AND
CompleteDescription.lid = Location.lid
```

• *SVM Approach*

– *Extension Intersection-based Approach*

* *Minimal Storage Policy* (I, \leq)

In this case, $V_i(st) = \{t \in N(st.ctx_i) \cup Inc_i(st.ctx_i) \mid \bar{I}(t) \cap st.A \neq \emptyset\} \cup B^*(st.ctx_i)$. Query:

```
SELECT DISTINCT(Location.lname)
FROM Location
WHERE Location.lid IN ((( $q_N^{(1)}$ (st.ctx_i) OR ( $q_N^{(1)}$ ( $q_B^{(d*)}$ (st.ctx_i)) MINUS st.ctx_i))
AND
(SELECT COUNT(*) FROM Hotel WHERE Hotel.lid in  $q_N^{(d*)}$ (Location.lid)
INTERSECT st.A ) > 0) OR  $q_B^{(d*)}$ (st.ctx_i))
```

* *Maximal Storage Policy* (\bar{I}, \leq^r)

Here, $V_i(st) = \{t \in N(st.ctx_i) \cup Inc_i(st.ctx_i) \mid \bar{I}(t) \cap st.A \neq \emptyset\} \cup B^*(st.ctx_i)$.

Query:

```
SELECT DISTINCT(Location.lname)
```

```

FROM Location
WHERE Location.lid IN ((( $q_N^{(1)}(st.ctx_i)$ ) OR ( $q_N^{(1)}(q_B^{(d*)}(st.ctx_i))$  MINUS  $st.ctx_i$ ))
AND
(SELECT COUNT(*) FROM CompleteDescription WHERE lid = Location.lid
INTERSECT  $st.A$  ) > 0) OR  $q_B^{(d*)}(st.ctx_i))$ 

```

– *Description-based Approach*

* *Minimal Storage Policy* (D, \leq)

Here, $V_i(st) = \{t' \in B^*(t) \mid t \in D(o), \forall o \in st.A\} \cap (N(st.ctx_i) \cup Inc_i(st.ctx_i) \cup B^*(st.ctx_i))$. Query:

```

SELECT DISTINCT(Location.lname)
FROM Hotel, Location
WHERE Hotel.hid IN  $st.A$  AND Location.lid IN
 $q_B^{(d*)}(q_L(Hotel.hid))$ 
INTERSECT
SELECT lname FROM Location WHERE
lid IN ( $q_N^{(1)}(st.ctx_i)$  OR ( $q_N^{(1)}(q_B^{(d*)}(st.ctx_i))$  MINUS  $st.ctx_i$ ) OR  $q_B^{(d*)}(st.ctx_i))$ 

```

* *Maximal Storage Policy* (\bar{D}, \leq^r)

In this case, $V_i(st) = \{t \in \bar{D}(o), \forall o \in st.A\} \cap (N(st.ctx_i) \cup Inc_i(st.ctx_i) \cup B^*(st.ctx_i))$. Query:

```

SELECT DISTINCT(Location.lname)
FROM Location, CompleteDescription
WHERE CompleteDescription.hid IN  $st.A$  AND
CompleteDescription.lid = Location.lid
INTERSECT
SELECT lname FROM Location WHERE
lid IN ( $q_N^{(1)}(st.ctx_i)$  OR ( $q_N^{(1)}(q_B^{(d*)}(st.ctx_i))$  MINUS  $st.ctx_i$ ) OR  $q_B^{(d*)}(st.ctx_i))$ 

```

3.3.2.10 Zoom-out points Computation

According to the section 3.2.3.2, we need to compute the zoom-out points only when we need to zoom-out from the current focus. In this case, as he have already described, we can execute

the same queries as in the zoom-in points computation.

3.3.2.11 Count Information

In case we follow the maximal storage policy, to provide count information for a zoom-in point we only need to add the *count(*)* function in the query. For example, in case of *EVM*, *maximal storage* policy and *Extension Intersection-based* evaluation approach the query would be:

```
SELECT DISTINCT(Location.lname),COUNT(*)
FROM Location
WHERE
(SELECT COUNT(*) FROM CompleteDescription WHERE lid = Location.lid
INTERSECT st.A ) > 0 GROUP BY Location.lname
```

3.4 Faceted Exploration User Interfaces

Faceted and dynamic taxonomies are used more and more nowadays in a plethora of application domains, and recently also in general purpose Web search engines⁵. There are already several applications of faceted metadata search in e-commerce (e.g. ebay), library and bibliographic portals (e.g. DBLP), museum portals (e.g. MuseumFinland [23]), mobile phone browsers (e.g. FaThumb[24]), yellow pages portals (e.g. Veturi [27]). There are also some attempts to apply this interaction paradigm over Semantic Web (e.g. [21, 29, 32]), as well as over general purpose web search engines (e.g. Google Base), and interaction frameworks (e.g. mSpace[44]). Below we will discuss some user interfaces from various implementations.

The Flamenco interface permits users to navigate by selecting from multiple facets [59]. The information base is specific and contains art, architecture, and tobacco documents. Additionally, It provides an additional service that gives user the ability to see the description of an object. When user put the pointer of his mouse over an object o s.t. $o \in \bar{I}(ctx)$ then the terms $t \in D_I(o)$ are highlighted. In Figure 3.7 the displayed images have been filtered by specifying values for two facets (*Materials* and *Structure Types*). The matching images are grouped by subcategories of the *Materials* facet's selected *Building Materials* category.

Sacco in [42] proposes faceted exploration interaction scheme for government e-services which are available to citizens. The information base of such services is so complex and the usage of

⁵e.g. Google Base (<http://base.google.com/>)

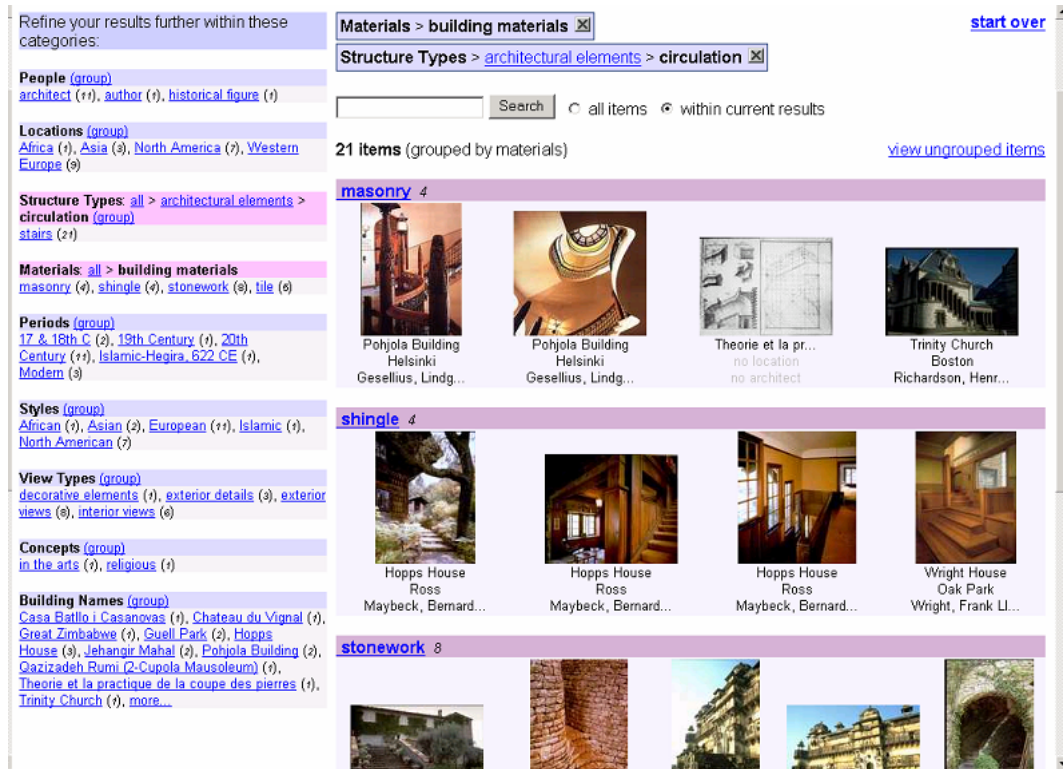


Figure 3.7: Flamenco User Interface

the specific interaction scheme make it easier. Figure 3.8 depicts the e-government portal which classifies all e-government information in 7 facets.

An other application domain of dynamic taxonomies is the Semantic Web. Searching for a specific information in Semantic Web is not easy as data have heterogenous character. /facet is a browser for Semantic Web developers as an instant interface to their complete dataset [21]. It gives user the ability to navigate through facets and make zoom operations based on properties. Additionally, /facet browser can handle any RDFS dataset without any additional configuration. Figure 3.9 shows /facet user interface.

At the same domain, MuseumFinland publishes heterogeneous museum collections on the Semantic Web [23]. It shares a set of ontologies and makes its rich collection semantically interoperable. The portal provides faceted exploration services over these collections (see Figure 3.10).

Fathumb is a novel approach for supporting faceted exploration services on hierarchical metadata from a mobile phone [24]. Figure 3.11(a) shows the user interface of Fathumb where Yellow Pages listings are described by 6 of 8 available facets e.g. *category*, *distance*, *location*, *hours*, *price* and *ratings*. Facets *favorites* and *shortcut* are inactive as their count information

SERVICES	LIFE EVENTS	TYPE	WHERE ARE YOU	CITIZENSHIP	SPECIAL RIGHTS	PROFILE
Income taxes (20)	Having a child (12)	offline service (75)	abroad (15)	Italian (500)	Women (35)	Sex (539)
Job search services (3)	Studying (35)	online service (375)	Italy (524)	EU (480)	Senior citizens (67)	Age (539)
Social security contributions (12)	Working (67)	guide (69)		extra-EU (56)	Handicapped (66)	Education (539)
Personal documents (7)	Transportation (43)				Relationships (43)	
Car registration (4)	Housing (30)					
Application for building permission (25)	Family (55)					
Declaration to the police (15)	Paying taxes (57)					
Public libraries (9)	Going abroad (12)					
Certificates (40)	Health (60)					
Enrollment in higher education / university (7)	Sport (255)					
Change of address (4)	Police (35)					
Health related services (60)	Leisure and culture (44)					
	Helping others (12)					
	Retiring (15)					

Figure 3.8: E-government portal with dynamic taxonomies

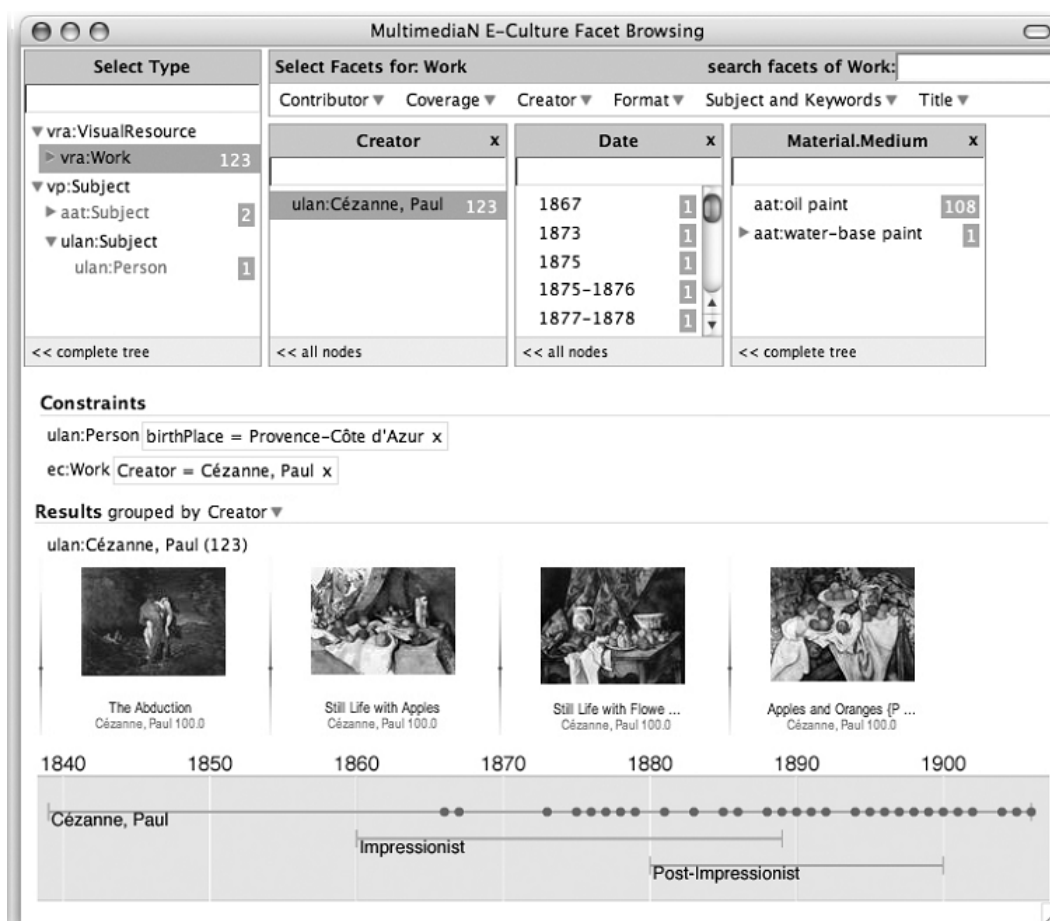





Figure 3.9: /facet User Interface

Uusi haku | Ohjeet | Näytä kaikki kategoriat | Tietoa ohjelmasta | MuseoSuomi-palaute

Sanahaku: Hee ☐ tarkenna hakua

Esinetyyppi [kaukku](#) > [työvälineet](#) (koko luokittelu)

[tekstiilikasityövälineet](#) (219),
[kansanlaakunnan työvälineet](#) (1),
[luokittelemattomat työvälineet](#) (36),
[maaloustyövälineet](#) (7), [metallityövälineet](#) (1),
[palkkomas ja hienontamisvälineet](#) (4),
[kirjoitusvälineet](#) (9), [metsätyövälineet](#) (4),
[työkälut](#) (22)

Materiaali (koko luokittelu) (ryhmittele kohteet)
[materiaalit](#) (241)

Valmistaja (koko luokittelu) (ryhmittele kohteet)
[henkilöt](#) (9), [tuotemerkit](#) (2),
[yritykset](#) (38)

Valmistuspaikka (koko luokittelu) (ryhmittele kohteet)
[Afrikka](#) (2), [Etelä-Amerikka](#) (1),
[Eurooppa](#) (84)

Valmistusaika (koko luokittelu) (ryhmittele kohteet)
[aikakaudet](#) (90), [vuosisadat](#) (89)

Käyttäjät (koko luokittelu) (ryhmittele kohteet)
[henkilöt](#) (54), [laitokset](#) (1),
[yritykset](#) (3)




Käyttöpaikka (koko luokittelu) (ryhmittele kohteet)
[Eurooppa](#) (71)

Käyttötilanne (koko luokittelu) (ryhmittele kohteet)
[harrastus- ja kansalaistoiminta](#) (4),
[kohteelle tehtävät toimenpiteet](#) (17),
[maalous ja karjanhoito](#) (2),
[ruoan- ja juomanvalmistus](#) (3),
[toimijoiden yleiset prosessit](#) (2),
[elinkeinot](#) (9), [valmistustekniikat](#) (179)

Kokoelma (koko luokittelu) (ryhmittele kohteet)
[Espoon kaupunginmuseon kokoelmat](#) (54),
[Kansallismuseon kokoelmat](#) (193),
[Lahden kaupunginmuseon kokoelmat](#) (50)

Hakuehdot
Kategoria: Esinetyyppi > [työvälineet](#) (ryhmittele kohteet) [goita](#)

Kohteet ryhmiteltyinä kategorian [työvälineet](#) mukaisesti
 (näytä ilman ryhmitelyä)
[tekstiilikasityövälineet](#), kohteet 1-4/219 (ryhmittele kohteet)

 kehräpuu, kuosali (NBA SU4527 50)	 kehrulauta, kehräpuu, kuezzel, kuosali (NBA SU5069 26)	 rukinlapa (ECM 100 1)	 sneidde, varttinänlumpio, varttinäpyörä (NBA SU2449 7) (edellinen) / (seuraava)
--	--	--	---

[kansanlaakunnan työvälineet](#), kohteet 1-1/1 (ryhmittele kohteet)

 suonirauta-suoneniskentärauta (ECM 2711 1)	(edellinen) / (seuraava)
---	--------------------------

[luokittelemattomat työvälineet](#), kohteet 1-4/36 (ryhmittele kohteet)




 nappikoukku-napituskoukku (ECM 3594 264)	 kietkamähdzi, komsiolahna (NBA SU4922 32)	 palohosat-palohosat (ECM 614 1)	 luontilasta (NBA SU4135 166)
---	---	--	---

Figure 3.10: Museum Finland User Interface

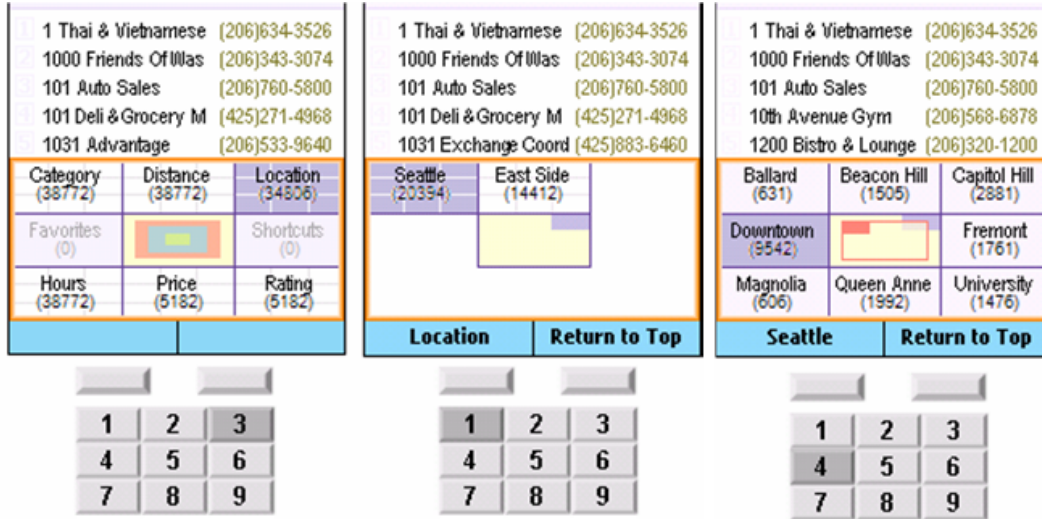


Figure 3.11: Fathumb User Interface

is zero. To zoom in a facet you have just to select the relative button from mobile's keyword. In Figure 3.9(a) user selects the facet *location*. Figure 3.9(b) shows the zoom-in points of the *location* facet. User has the ability to zoom-in or go back to his previous selection (zoom-out) or return to top in order to select an other facet (see at the bottom of mobile screen in Figure 3.9(b),(c)).

An other faceted exploration system on Yellow pages described in [27]. The yellow pages service portal Veturi contains some 220,000 real-world services. The user interface of Veturi is based on on-the-fly semantic autocompletion of keywords into categories, made possible by the use of AJAX⁶ techniques. Figure 3.12 depicts the search interface of the Veturi portal where a user trying to find out where he can buy rye bread in Helsinki. He has already selected Helsinki as the locale for the services he requires and he is in the process of describing the actual service.

Finally, a faceted exploration interface that use more and more the academic community is the Faceted DBLP⁷. User can search publications by author or venue and the answer of his query are loaded to the faceted search system. Faceted search engine provides 4 facets: venue, author, year, publication type. Figure 3.13 depicts Faceted DBLP interface.

⁶Asynchronous Javascript And XML

⁷http://dblp.l3s.de/?q=&newQuery=yes&resTableName=query_result9prpDC

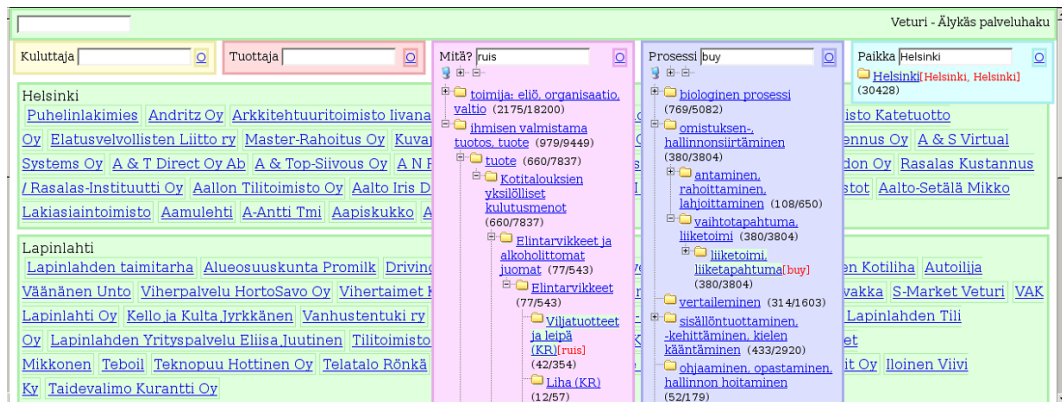


Figure 3.12: Veturi User Interface

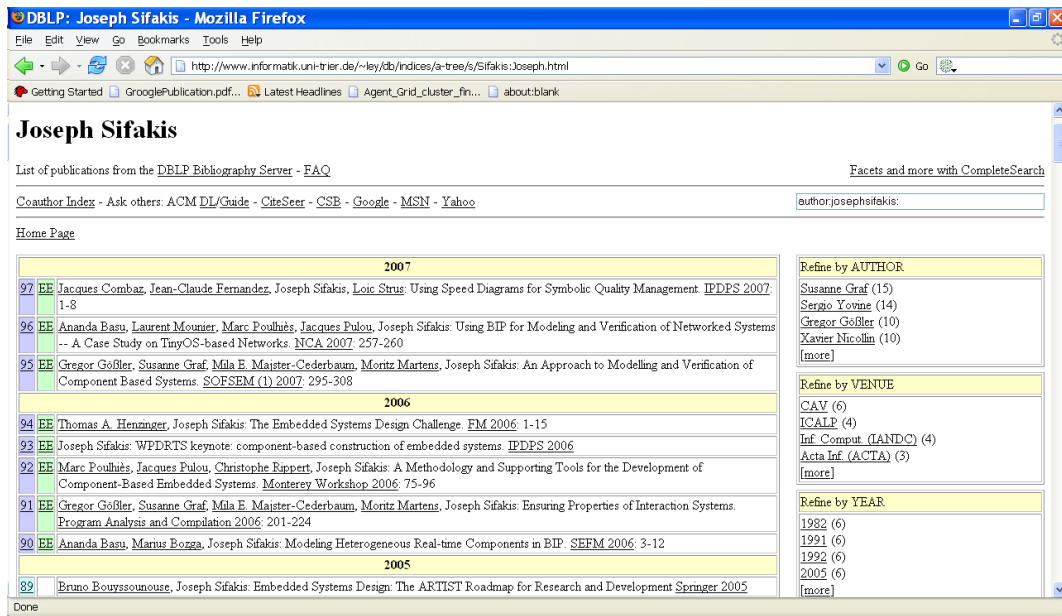


Figure 3.13: DBLP User Interface

3.5 FDT in Commercial Web-sites

Section 3.5.1 presents a number of commercial faceted metadata search engines while section 3.5.2 describes an open XML specification for defining and sharing faceted classification schemes.

3.5.1 Commercial Faceted Metadata Search Engines

Table 3.3 lists a number of faceted metadata search engines and for each one of them some commercial sites in which they are used.

Metadata Search Engines	Used in commercial sites	Support of Zoom-in Points	Support of count information	Support over DBMS	Support of remote sources	I/O Formats	Other supported features
Knowledge Processors ⁸	Non Commercial: tiziano.di.unito.it, erare.di.unito.it	Yes	Yes	No	Yes		rule-base autotclassifier for XML sources, integrated IR component
CAMELIS	personal data	Yes	Yes	No	Yes (URLs)	CSV, JPEG, MP3, BIBTEX	automatic and manual classification, querying by examples, export of playlists and slideshows
i411	ElectionsOntario.on.ca, DeTele-foongids.nl, iLocal.net	Yes	Yes	Yes	Yes	HTML, XML, PDF, DOC, PPT, XLS, ...	Predefined taxonomies and categories, compressed on-disk storage.
Mercado	Blockbuster.com, Sears.com, USOPNet.com, officemax.com	Yes	Yes	Yes	Yes	Integrates with external personalization systems.	
Siderean Seamark	Indiana Educational Clearinghouse, Fortunoff.com, Environmental-HealthNews.org	Yes	Yes	Yes	Yes	XML, RDF, RSS, flat files.	Predefined taxonomies and categories, uses RDF an intermediate storage format.
Endeca	TowerRecords.com, BarnesAnd-Noble.com, Spiegel.com, Cabot-Corp.com	Yes	Yes	Yes	Yes	XML database imports	Predefined taxonomies and categories
Solr	Repubblica.it, StubHub.com, Archive.com, Chowhound.com, CNet.com	Yes	Yes	Yes	Yes	HTML, OpenOffice, DOC, XLS, PPT, IMAP, RTF, PDF, etc.	It uses the Lucene Search Library and extend it.
Google Base	base.google.com	Yes	No	No	Yes	PDF, XLS, TXT, HTML, RTF, WPD, ASCII, XML	

Table 3.3: Faceted Metadata Search Engines in commercial sites

```

<?xml version="1.0" ?>
<xfml version="1.0" url="http://domain.com/xfml/map1.xml" language="en-us">
  <facet id="from">By From</facet>
  <topic id="eu" facetid="from"><name>Europe</name></topic>
  <topic id="it" facetid="from" parentTopicid="eu"><name>Italy</name></topic>
  <topic id="gr" facetid="from" parentTopicid="eu"><name>Greece</name></topic>
  <topic id="ro" facetid="from" parentTopicid="it"><name>Rome</name></topic>
  <topic id="mi" facetid="from" parentTopicid="it"><name>Milan</name></topic>
  <topic id="ma" facetid="from" parentTopicid="mi"><name>Malpensa</name></topic>
  <topic id="li" facetid="from" parentTopicid="mi"><name>Linate</name></topic>
  <topic id="at" facetid="from" parentTopicid="gr"><name>Athens</name></topic>
  <topic id="cr" facetid="from" parentTopicid="gr"><name>Crete</name></topic>
  <topic id="ch" facetid="from" parentTopicid="cr"><name>Chania</name></topic>
  <topic id="ir" facetid="from" parentTopicid="cr"><name>Iraklio</name></topic>
  <facet id="to">By To</facet>
  ....
  <facet id="date">By Date</facet>
  ....
  <facet id="price">By Price</facet>
  ....
  <page url="http://flight1.com/">
    <title>Flight1</title>
    <description>Flight 1 from Malpensa to Heraklion</description>
    <occurrence topicid="ma" />
    ....
  </page>
</xfml>

```

Figure 3.14: XFML file example

3.5.2 XFML

eXchangeable Faceted Metadata Language (XFML) is an open XML specification for defining and sharing faceted classification schemes [2]. It provides a simple format to share classification and indexing data. The building blocks of a faceted hierarchy in XFML are facets and concepts (or topics). A facet is the top node of each tree. The nodes in the tree are called topics. XFML can define multiple hierarchies, and each hierarchy is a facet.

The `<facet id="from">By From</facet>` tag denotes the facet with name "By From" while the `<topic id="it" facetid="from" parentTopicid="eu"><name>Italy</name></topic>` tag denotes the term $Italy \in T_{ByFrom}$ as *facetid* property defines the facet that term belongs and $Italy \in N(Europe)$ as *parentTopicid* parameter denotes his parent. The `<page>` tag denotes an object while `<occurrence>` tags denote the $D_I(o)$. Figure 3.14 depicts the materialized faceted taxonomy of our running example expressed in XFML.

There are not a lot of tools that support this standard. Drupal⁹ is a content management system which gives user the ability to export XFML. Facetmap¹⁰ lets you import and browse

⁹<http://www.drupal.com>

¹⁰<http://www.facetmap.com>

XFML files, and Taxomita¹¹ is an authoring tool built around XFML.

¹¹<http://www.taxomita.com>

Chapter 4

flexplorer & Applications

’Όσα με τη λογική βρίσκεις σωστά, αυτά εφάρμοσε τα και στη πράξη.

’Όσα δεν πρέπει να κάνεις, ούτε να τα σκέφτεσαι.’

Ισοκράτης (436 π.Χ. - 338 π.Χ.)

This chapter presents and describes an API for providing faceted exploration services. Furthermore, it presents various implemented applications and experimental results. In more details, Section 4.1 presents **flexplorer**, a main memory API for supporting faceted exploration. Section 4.2 and 4.3 present two applications of **flexplorer** on web search engines. Finally, 4.4 reports experimental results on DB-R architecture.

4.1 flexplorer API

flexplorer is a main memory API for providing faceted exploration services [56]. **flexplorer** follows the *minimal storage* approach (I, \leq), supports both *EVM* and *SVM* visualization modes, and *Extension Intersection-based* computational approach. Moreover, it supports both *click(t)* and *feed(A)* operations.

4.1.1 Specifications

flexplorer allows managing (creating, deleting, modifying) terms, taxonomies, facets and object descriptions. It supports both finite and infinite terminologies (e.g. numerically-valued attributes). In addition it supports explicitly and intentionally defined taxonomies. Examples

of the former include classification schemes and thesauri, while examples of the latter include hierarchically organized intervals (based on the *cover* relation).

The implementation is in Java, so the predefined ordering of built-in types (e.g. of `int`, `float`, `String`), as well as the customized ordering defined for user-defined Java classes (e.g. through the `comparable` interface) is exploited. To allow intentionally defined partially ordered domains, a `partiallyComparable` interface has been introduced and can be used by the developer. The framework also supports parametric types. Subsequently, regarding internal architecture, for each term $t \in \mathcal{T}$, `flexplorer` stores the $I(t)$ and the $N(t)$.

Regarding, interaction, the framework provides methods for setting (resp. computing) the focus (resp. zoom-in points). In addition, the framework allows materializing on demand the relationships of a taxonomy, even if the domain is infinite and intentionally defined (e.g. between numbers, intervals, etc), as this can accelerate the computation of $N(t)$ at the cost of extra main memory space (to keep the relationships). Regarding deployment, the framework can be used either at the server side or at client side, depending on the case. Finally, the results can be loaded on `flexplorer` in 4 different ways: JDBC ResultSet, XML file, TXT file and ResultDocument (is an Object that have been defined in API).

In more detail, `flexplorer` supports:

- **State Transitions**

- $click(t)$, by defining term's name
- $click(t)$, by defining term's id
- $click(t)$, by defining term's taxonomy path
- $feed(A)$
- boolean expressions which allow multiple $click(t)$ operations

- **Facets' Structures**

- finite and infinite terminologies
- explicitly and intentionally defined taxonomies
- intentionally defined partially ordered domains
- DAGs

- **Loading / Storing**

- loading objects from JDBC ResultSet
- loading objects from XML files
- loading objects from TXT files
- loading objects as ResultDocument (is an Object that has been defined in API)
- loading/storing faceted taxonomies in TXT and XML files under specific format

- **Other Functionalities**

- checks for redundant relationships
- checks for cycles
- zoom-in points ranking by: (a) count descending, (b) count ascending, (c) alphabetical order descending and (c) alphabetical order ascending

The official website of **flexplorer** is: <http://www.ics.forth.gr/~tzitzik/flexplorer/index.html>

4.1.1.1 Class Diagrams

Figure 4.1 presents the class diagram of the API. The main packages of the API are described in more details in the Figures 4.2 - 4.7.

API contains 13 packages. They are described below:

1. **Terms:** The classes of this package implement the entity Term. The objects are classified under a Term. A Term in the UI layer is a zoom point.
2. **Terminologies:** This package manages the functionality of a facet's terminology, where terminology = a list of Terms.
3. **Taxonomies:** This package implements the relationships between the terms.
4. **Facets:** A facet has a Taxonomy and a name.
5. **FacetedTaxonomies:** A faceted taxonomy contains a list of facets.
6. **MaterializedFacetedTaxonomies:** This package implements the entity: materialized faceted taxonomy. A materialized faceted taxonomy contains a faceted taxonomy and the

objects which are classified under the terms of the faceted taxonomy. The classes of this package provide methods for defining the focus, computing the legal objects with respect to the focus, computing the new zoom-points and their counts etc.

7. **Types:** The framework supports parametric types, this package defines these parametric types.
8. **Comparators:** This package contains various comparators.
9. **IndexesSetting:** As we already described in 4.1.1, `flexplorer` can load objects with 4 different ways. The classes of this package supports this functionality.
10. **Storing:** The classes of this package support the storing of a materialized faceted taxonomy in txt and xml files.
11. **Resources:** This package contains a class named Resources. This class contains various resources for the API such as the global name of the \mathcal{T} .
12. **Util:** It contains various utile classes and methods.
13. **BooleanParser:** This package contains classes which implement a boolean parser.

4.1.2 An Example of Using the API

In this section we present an example of using the `flexplorer` API for constructing a materialized faceted taxonomy that consists of two facets (one flat and one hierarchically organized) and three classified objects. The example shows two ways for setting the focus.

```

1 public class DemoAPIClient {

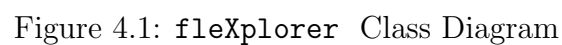
    public DemoAPIClient() {}

    public static void main(String[] args) {
6        // Creates a new materialized faceted taxonomy
        MFTMEM materializedTaxonomy = new MFTMEM("Mitos DB", Counters.TRUE, ObjectFacet
            .YES);

        //Creates the faceted taxonomy
11       FT facetedTaxonomy = new FT();

        // Filetypes ' Taxonomy

```



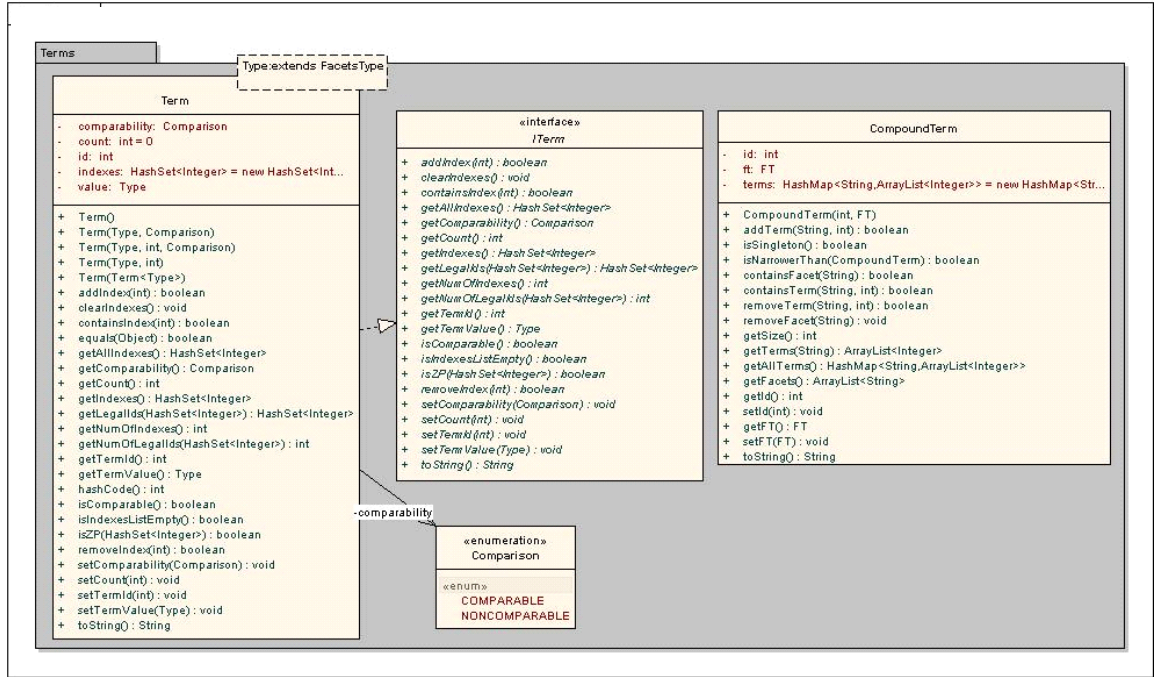


Figure 4.2: Terms' Package Class Diagram

```

Taxonomy<StringType> filetypeTaxonomy = new Taxonomy<StringType>("Filetypes",
    CheckRel.FALSE, Comparison.NONCOMPARABLE, HasLists.FALSE, "String");
// Domains' Taxonomy
Taxonomy<StringType> domainsTaxonomy = new Taxonomy<StringType>("Domains",
    CheckRel.FALSE, Comparison.NONCOMPARABLE, HasLists.TRUE, "String");

// Creates the relationships of filetype taxonomy. The facet is flat.
filetypeTaxonomy.setTerm(new StringType("pdf"));
filetypeTaxonomy.setTerm(new StringType("doc"));
filetypeTaxonomy.setTerm(new StringType("txt"));

// Creates the relationships of domains taxonomy. The facet is hierarchically
    organized.
domainsTaxonomy.addHead(new StringType("gr"));
domainsTaxonomy.setRelship(new StringType("gr"), new StringType("uoc.gr"));
domainsTaxonomy.setRelship(new StringType("gr"), new StringType("forth.gr"));
domainsTaxonomy.setRelship(new StringType("uoc.gr"), new StringType("csd.uoc.gr"));
;
domainsTaxonomy.setRelship(new StringType("forth.gr"), new StringType("ics.forth.
    gr"));

// Add taxonomies to facets.
Facet<StringType> facet1 = new Facet<StringType>("By filetype", filetypeTaxonomy,
    IndexType.SIMPLE);

```

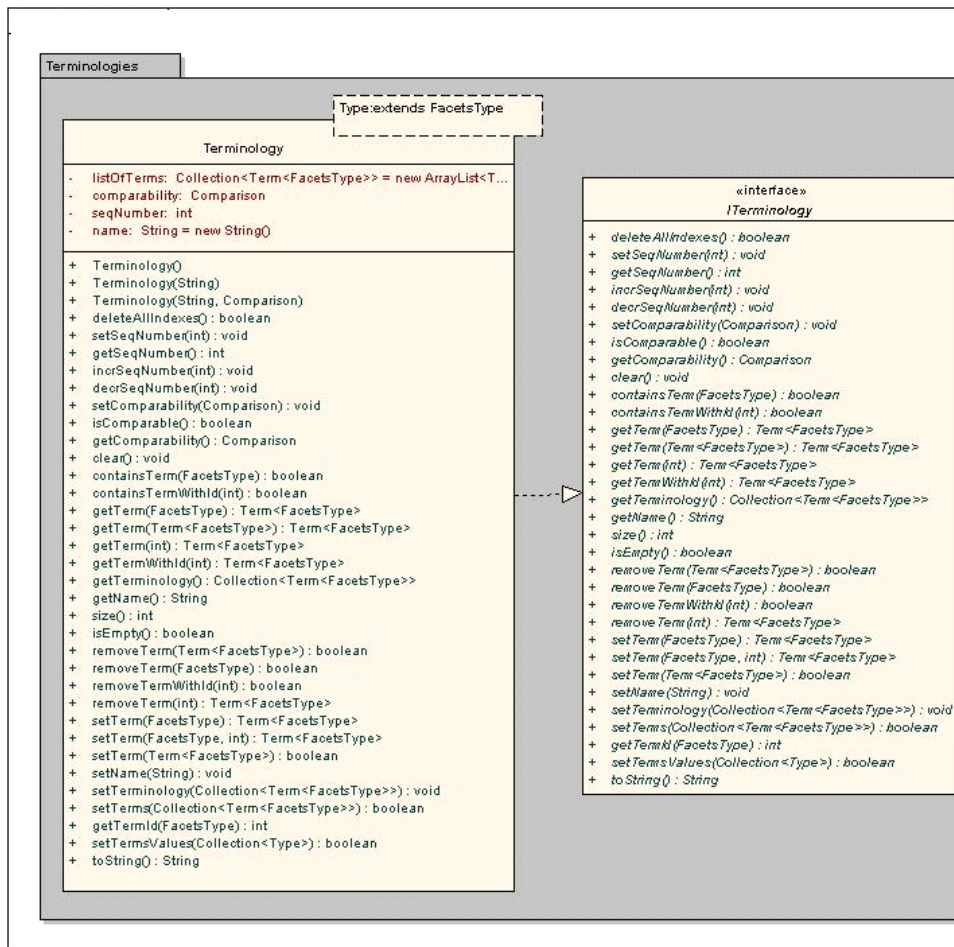


Figure 4.3: Terminologies' Package Class Diagram

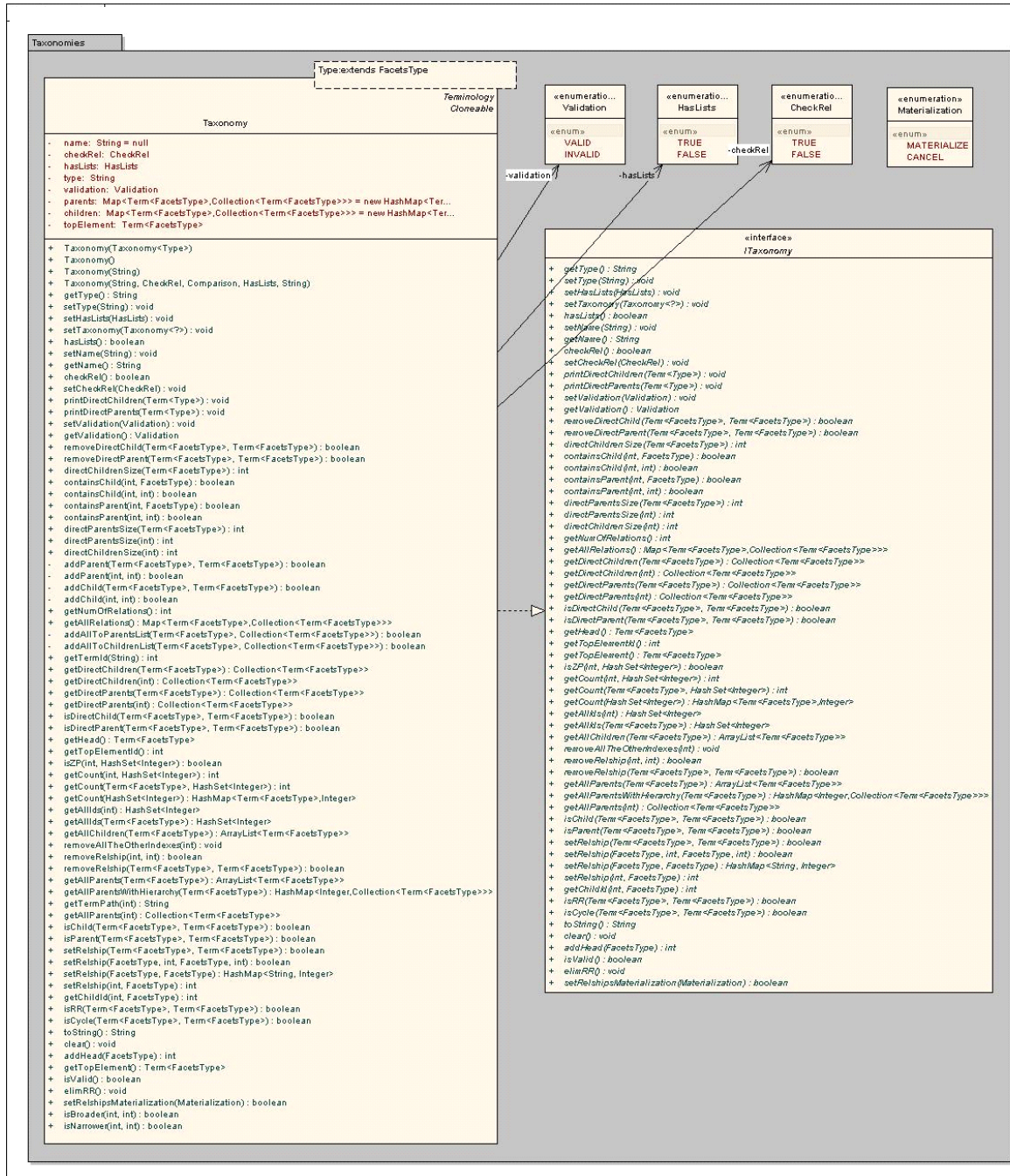


Figure 4.4: Taxonomies' Package Class Diagram

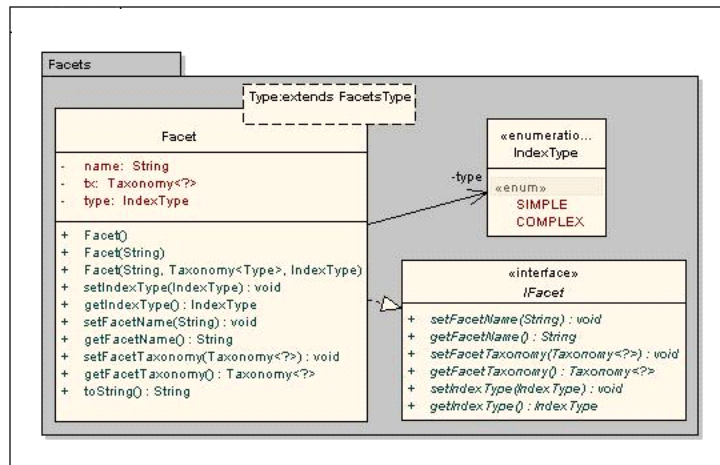


Figure 4.5: Facets' Package Class Diagram

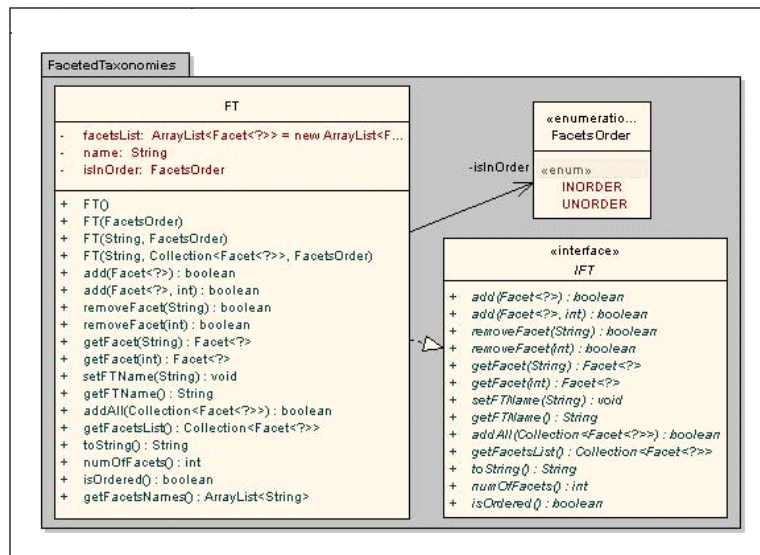


Figure 4.6: Faceted Taxonomies' Package Class Diagram

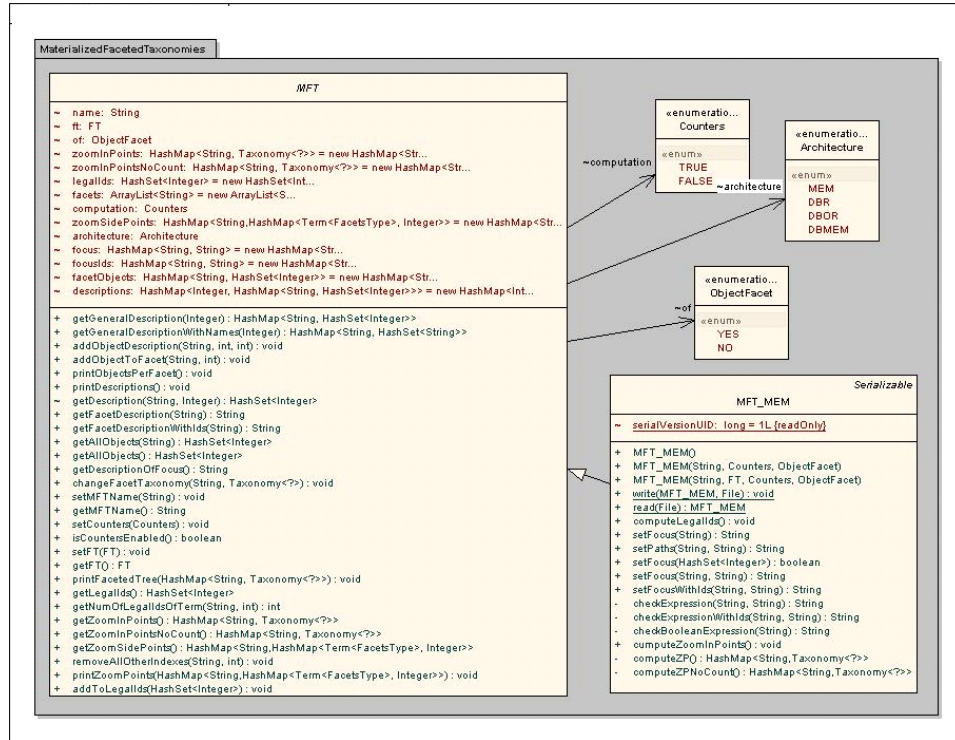


Figure 4.7: Materialized Faceted Taxonomies' Package Class Diagram

```

Facet<StringType> facet2 = new Facet<StringType>("By domain", domainsTaxonomy,
IndexType.SIMPLE);

// Add facets to facetTaxonomy
facetedTaxonomy.add(facet1);
facetedTaxonomy.add(facet2);

// Add facet taxonomy to materialized faceted taxonomy
materializedTaxonomy.setFT(facetedTaxonomy);

// Each object is a Document. This list contains all the objects.
ArrayList<Document> docs = new ArrayList<Document>();

// This HashMap contains the term of each facet that a Document is classified
under.
HashMap<String, FacetsType> st1 = new HashMap<String, FacetsType>();
st1.put("By domain", new StringType("ics.forth.gr"));
st1.put("By filetype", new StringType("doc"));

// This Document has the id 1, ranking 1 and is classified under the term ics.
// forth.gr of
// facet By domain and under the term doc of the facet By filetype.
Document doc1 = new Document(1,1,st1);

```



```

51 docs.add(doc1);

HashMap<String, FacetsType> st2 = new HashMap<String, FacetsType>();
st2.put("By domain", new StringType("ics.forth.gr"));
st2.put("By filetype", new StringType("txt"));
56 Document doc2 = new Document(2,2,st2);
docs.add(doc2);

HashMap<String, FacetsType> st3 = new HashMap<String, FacetsType>();
st3.put("By domain", new StringType("csd.uoc.gr"));
61 st3.put("By filetype", new StringType("doc"));
Document doc3 = new Document(3,3,st3);
docs.add(doc3);

// Creates the extensions of each term.
66 Documents dts = new Documents(materializedTaxonomy, docs);
dts.setIndexes();

// Computes the ids that belong to the extension of the focus.
// At first the focus is the top element of each facet.
71 materializedTaxonomy.computeLegalIds();

// Computes the zoom-in points
materializedTaxonomy.computeZoomInPoints();

76 // Gets the zoom-in points
HashMap<String, Taxonomy<?>> tmp = materializedTaxonomy.getZoomInPoints();

// Prints the facets.
materializedTaxonomy.printFacetedTree(tmp);
81

// Prints the focus.
System.out.println(materializedTaxonomy.getDescriptionOfFocus());

// Sets the focus. User prefers the documents which are
86 // classified under term csd.uoc.gr of facet By domain.
materializedTaxonomy.setFocus("By domain", "csd.uoc.gr");
materializedTaxonomy.computeLegalIds();
materializedTaxonomy.computeZoomInPoints();
tmp = materializedTaxonomy.getZoomInPoints();
91 materializedTaxonomy.printFacetedTree(tmp);

// Prints the ids of the legal Documents wrt the focus in our case the id: 3
System.out.println("LegalIds: "+materializedTaxonomy.getLegalIds());

96 // Set the focus with a boolean expression.

```

```

101     materializedTaxonomy.setFocus("{By filetype : doc OR txt} AND {By domain : ics.
        forth.gr}");
    materializedTaxonomy.computeLegalIds();
    materializedTaxonomy.cumputeZoomInPoints();
    tmp = materializedTaxonomy.getZoomInPoints();
    materializedTaxonomy.printFacetedTree(tmp);

    System.out.println(materializedTaxonomy.getDescriptionOfFocus());
    System.out.println("LegalIds: "+materializedTaxonomy.getLegalIds());
106 }

```

4.1.3 Desktop-based Client

In this section, we present a Desktop-based graphical client of **flexplorer** for supporting faceted exploration. In more details, it loads the facets and the records (using specific configuration files) to the main memory and provides faceted exploration services.

Figure 4.8 presents the welcome screen of application. By pressing the "next" button, the screen of Figure 4.9 will be presented. Here the user has to specify the configuration file for facets. If user selects the configuration file correctly and presses the "next" button a same screen will be presented for loading the configuration file of objects. Finally, by pressing the "finish" button, the objects will be loaded to the main memory successfully and the faceted exploration user interface of Figure 4.10 will be presented.

4.1.4 Experimental Evaluation

Below we report some experimental results using the **flexplorer**.

We created a materialized faceted taxonomy consists of 4 facets, each was a balanced and complete tree with degree = 3 and depth = 5. A dataset of 10^6 objects was created where each object was classified under a randomly selected term from each facet.

Firstly, we measured the time **flexplorer** needs to load a number of object (or the results which belong to the answer of a query on a web search engine). Figure 4.11 shows the loading time of the top-L answer to the **flexplorer** for 4 different methods as we have already described in 4.1. The loading time for each answer size and for each different method is the average time of 10 executions. It is obvious that the ResultDocument is the faster one since the data are kept

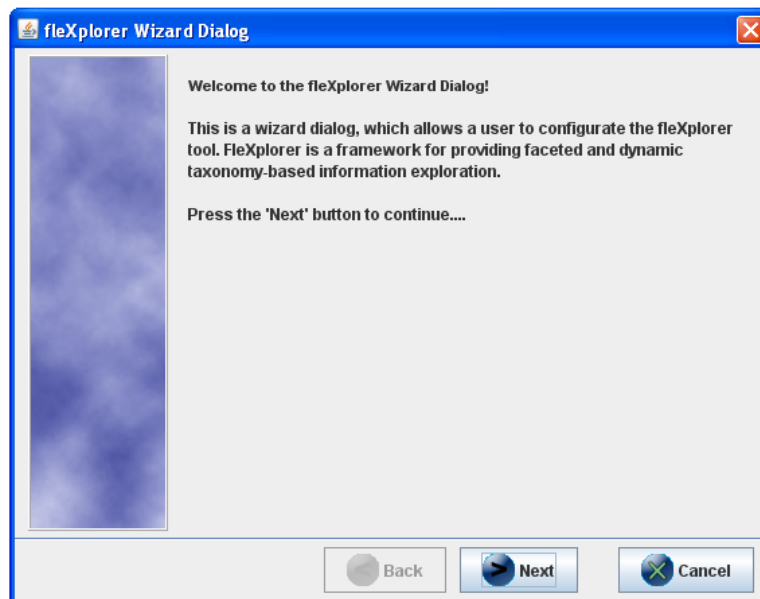


Figure 4.8: Desktop-based Client: Welcome Screen

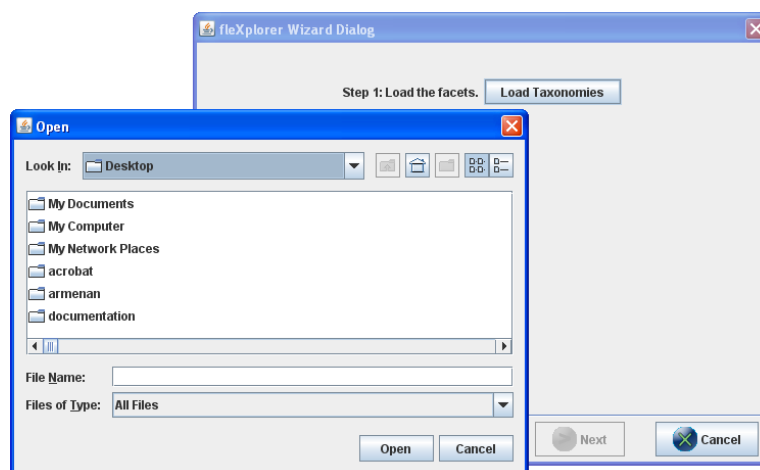


Figure 4.9: Desktop-based Client: Facets and Objects Loading

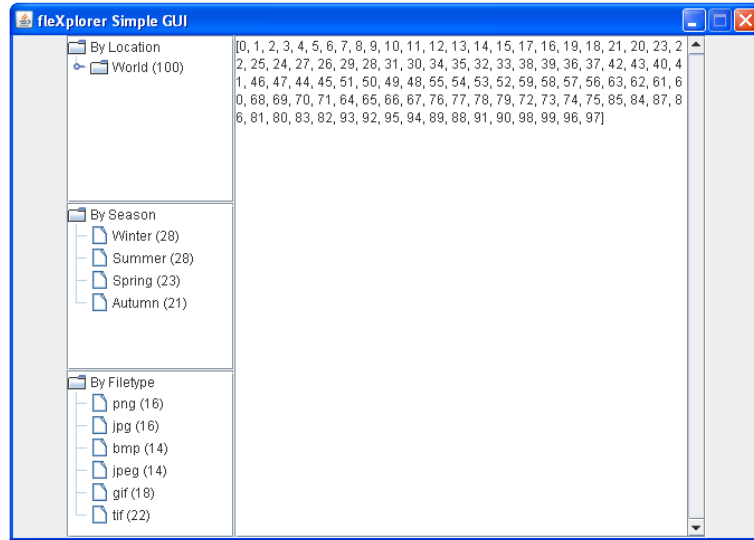


Figure 4.10: Desktop-based Client: Faceted Exploration UI

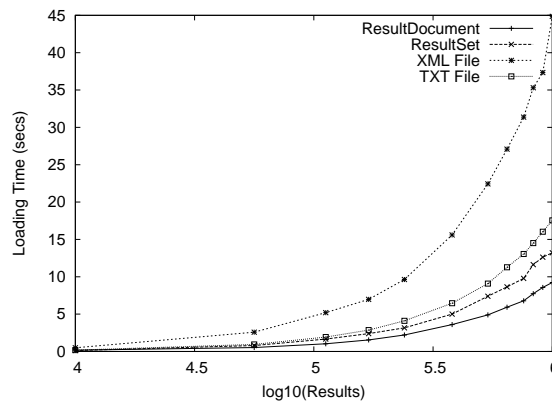


Figure 4.11: Time to load results to flexplorer

in main memory, followed by the ResultSet times, since in this case the data are stored in the database. Parsing a txt file with predefined data locations is much slower than the above and parsing the xml file is the slowest one. For the experiments we used a Pentium IV 3GHz with 2GB RAM and using Windows XP.

Figure 4.12 shows the time to compute the zoom-in points for the above four facets after the selection of a zoom-in point, with and without count information. Each time is the average time of 20 executions (5 executions for every results loading way). It is obvious that computing zoom-in points with count information is more expensive than not computing it. From the figure above, in 1 sec we can compute the zoom-in points of 240.000 results with count information and 540.000 without.

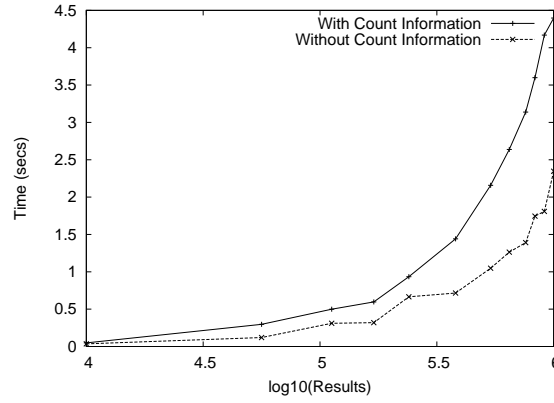


Figure 4.12: Time to compute zoom-in points

4.2 Application on a Web Search Engine

As we already described in previous chapters, faceted and dynamic taxonomies are used more and more nowadays in a plethora of application domains. In this section we describe a web based application of `flexplorer` API.

4.2.1 Mitos WSE

`Mitos` [35, 34] (formerly known as `grOOGLE`)¹ is a prototype Web search engine that is being developed by the Department of Computer Science of the University of Crete and FORTH-ICS. `flexplorer` is used by `Mitos` for offering general purpose browsing and exploration services. On the basis of the top- L answer of each submitted query, the following four facets are created and offered to users:

- web domain, a hierarchy is defined (e.g. *csd.uoc.gr* < *uoc.gr* < *gr*),
- file type (e.g. pdf, html, doc, etc), no hierarchy is created in this case
- language of a document based on the encoding of a web page and
- (modification) date hierarchy.

Notice that each page in the top- L answer is (straightforwardly) classified to one term of each of the above taxonomies.

In more detail, when the user executes a query, `Mitos` returns an ordered list with the ids (wrt the ranking) of the documents which belong to the answer. Then, `Mitos` loads to

¹<http://google.csd.uoc.gr:8080/mitos/>

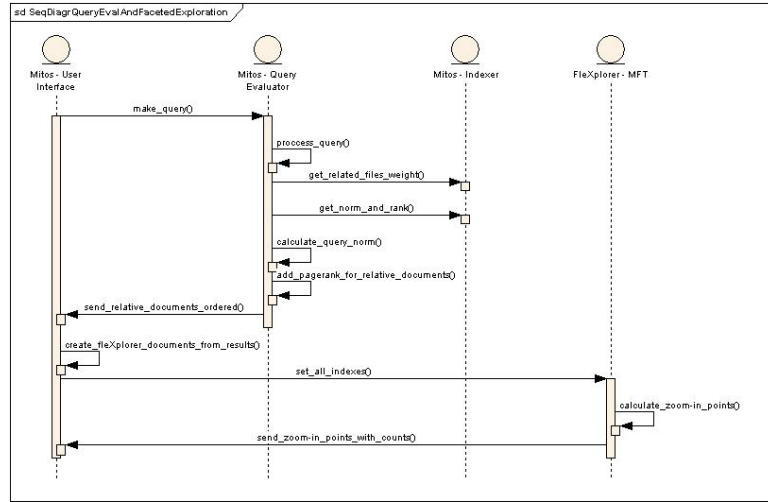


Figure 4.13: Mitos & flexplorer Sequence Diagram

flexplorer each document with the information for each facet. The hierarchies for web domain, file type and language are predefined, while the hierarchy of dates can be constructed with two different ways. It can be predefined e.g. from 1990 to 2010, or it can be constructed on-the-fly from the loaded documents. **flexplorer** provides functions for constructing automatically hierarchies from the $\bar{D}(o)$. Figure 4.13 presents a sequence diagram of **flexplorer** and **Mitos** communication.

After query execution and facets construction, user can start exploring the query answer. The facets are appeared at the right of the screen. To zoom-in, user has to select the zoom-in points by checking their checkboxes and presses the "OK" button. In more detail, after selecting the zoom-in points, user must select the "OR" or "NO" radio button. In case of "OR", the focus answer will consist of objects which belong to the union of the extensions of all selected zoom-in points. On the other hand, in case of "NO", the focus answer will contain the objects that do not belong to the extensions of the selected zoom-in points. In case we have multi classification for a facet, an "AND" radio button will be appeared. By selecting this, the focus answer will contain only the objects which belong to the intersection of the extensions of all zoom-in points. Figure 4.14 depicts the user interface of **Mitos**.

Furthermore, if the mouse be over a result then a box which will contain the complete descriptions of the document will be appeared at the right of the screen. For example, in Figure 4.14 we can see the complete descriptions for the first document.



Figure 4.14: Mitos user interface: Interpretations, Descriptions

Moreover, Mitos provide several services as regards the setting of the focus. After a zoom-in/out point execution, user can see and edit the current focus. Under the facets, a specific box which contain the focus is appeared. User can see the current focus, edit it and submit it. The focus has a specific format and supports boolean expressions. Figure 4.15 presents the specific service.

As the screen of a computer or a PDA may be too small to show all the facets (like the previous screenshot), or the user is not interested in a specific facet, or prefers a specific ranking of the facets, we have developed a mechanism which allows the user to drag and drop the facets (so that to place them in the desired order) and open or close them. Figure 4.16 shows the modified version of the GUI of Figure 4.17 where the user has opened only the facets *By Filetype* and *By Language* and prefers to show first the facet *By Filetype*. User's preferences are stored to the cache of the internet browser as cookies to persist over time.

4.2.2 Exploratory web searching with dynamic taxonomies and results clustering

Web Search Engines (WSEs) typically return a ranked list of documents that are relevant to the query submitted by the user. For each document, its title, URL and *snippet* (fragment of the text that contains keywords of the query) are usually presented. It is observed that most users



Figure 4.15: Mitos user interface: Focus

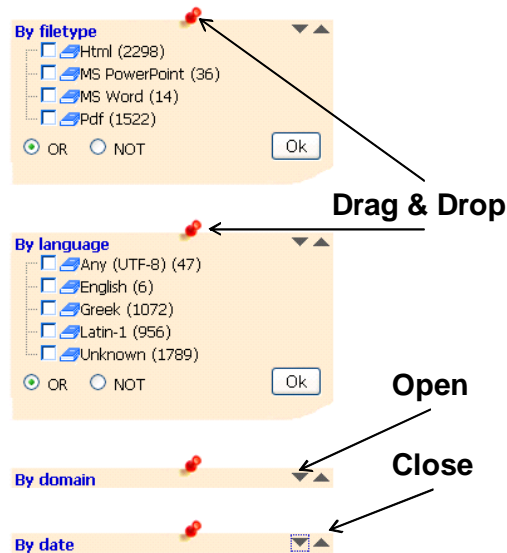


Figure 4.16: Modified Faceted Exploration UI according user's preferences

are impatient and look only at the first results [3]. Consequently, when either the documents with the intended (by the user) meaning of the query words are not in the first pages, or there are a few dotted in various ranks (and probably different result pages), it is difficult for the user to find the information he really wants. The problem becomes harder if the user cannot guess additional words for restricting his query, or the additional words the user chooses are not the right ones for restricting the result set.

One solution to these problems is *results clustering* [60] which provides a quick overview of the search results. It aims at grouping the results into topics, called *clusters*, with predictive names (labels), aiding the user to locate quickly documents that otherwise he wouldn't practically find especially if these documents are low ranked (and thus not in first result pages).

Another solution as we already discussed is to exploit the various static metadata that are available to WSEs in the context of the interaction paradigm of *faceted and dynamic taxonomies (FDT)*.

There are a few works in the literature [3, 26] that compare automatic results clustering with guided exploration (through FDT). In [33] we proposed combining these two approaches. In a nutshell, this work lies in: (a) proposing and motivating the need for exploiting both (static) explicit and (dynamically) mined metadata during Web searching, (b) showing how automatic results clustering can be combined with the interaction paradigm of dynamic taxonomies, by clustering on-demand the top elements of the user focus, (c) providing incremental evaluation algorithms for speeding up the interaction, and (d) reporting experimental results that prove the feasibility and the effectiveness of the approach.

To the best of our knowledge, there are no other WSEs that offer the same kind of information/interaction. A somehow related interaction paradigm that involves clustering is Scatter/-Gather [10, 20]. This paradigm allows the users to select clusters, subsequently the documents of the selected clusters are clustered again, the new clusters are presented, and so on. This process can be repeated until individual documents are reached. However, for very big answer sets, the initial clusters apart from being very expensive to compute on-line, will also be quite ambiguous and thus not very helpful for the user. Our approach alleviates this problem, since the user can restrict his focus through the available metadata, to a size that allows deriving more specific and informative cluster labels.

4.2.2.1 Coupling Static and Dynamically-mined Metadata for Exploration

FDT-based interaction is feasible for hundreds of thousands of objects very fast. However, the application of results clustering on thousands of snippets would have the following shortcomings:

- (a) *Inefficiency*, since real-time results clustering is feasible only for hundreds of snippets, and
- (b) *Low cluster label quality*, since the resulting labels would be too general. To this end we propose a *dynamic (on-demand) integration* approach. The idea is to *apply the result clustering algorithm only on the top- C (usually $C < 500$) snippets of the current focus and to redo this whenever the focus changes*. This approach not only can be performed fast, but it is expected to return more informative cluster labels. Let q be the user query and let $Ans(q)$ be the answer of this query. We shall use A_f to denote top- K (usually $K < 10^4$) objects of $Ans(q)$ and A_c to denote top- C objects of $Ans(q)$. Clearly, $A_c \subseteq A_f \subseteq Ans(q)$. Table 4.1 lists all symbols that are used in the sequel.

Symbol	Meaning
q	Current query
$Ans(q)$	Answer of the query as returned by the WSE
C	Number of top elements of the answer that will be clustered (usually $C < 500$)
K	Number of top elements of the answer that will be loaded to flexplorer (usually $K < 10^4$)
A_f	Top- K elements of $Ans(q)$
A_c	Top- C elements of $Ans(q)$
c_i	Cluster c_i
$Ext(c_i)$	Documents that belong to cluster c_i

Table 4.1: Table of Symbols

The steps of the process are the following:

- (1) The snippets of the elements of A_c are generated.
- (2) Clustering is applied on the snippets of the elements of A_c , generating a cluster label tree clt .
- (3) The set of A_f (with their metadata), as well as clt , are loaded to **flexplorer**, a module for creating and managing the FDT. As the facet that corresponds to automatic clustering includes only the elements of A_c , we create an additional artificial cluster label, named "REST" where we place all objects in $A_f \setminus A_c$ (i.e. it will contain $K - C$ objects).
- (4) **flexplorer** computes and delivers to the GUI the (immediate) zoom points.

The user can start browsing by selecting the desired zoom point(s), refining in this way the answer set. When a zoom point is selected the new focus is computed and steps (1)-(4) are performed again over the new A_f (and A_c). A more efficient incremental approach is described below.

4.2.2.2 Incremental Algorithm for Exploration

Here we present an incremental approach for exploiting past computations and results. Specifically the algorithm aims at reusing the snippets and their suffixes since this is the more expensive task of online results clustering.

Let A_f be the objects of the current focus. If the user selects a zoom point he moves to a different focus. Let A'_f denote the top- K elements of the new focus, and A'_c the top- C of the new focus. The steps of the algorithm follow.

- (1) We set $A_{c,new} = A'_c \setminus A_c$ and $A_{c,old} = A_c \setminus A'_c$, i.e. $A_{c,new}$ is the set of the new objects that have to be clustered, and $A_{c,old}$ is the set of objects that should no longer affect clustering.
- (2) The snippets of the objects in $A_{c,new}$ are generated (those of $A_{c,old}$ are available from the previous step). Recall that snippet generation is expensive.
- (3) Clustering is applied *incrementally* to $A_{c,new}$.
- (4) The new cluster label tree clt' is loaded to **flexplorer**.
- (5) **flexplorer** computes and delivers to the GUI the (immediate) zoom points for the focus with contents A'_f .

4.2.2.3 Implementation

The implementation was done in the context of **Mitos**, while **flexplorer** is used for offering general purpose browsing and exploration services. Figure 4.17 shows a screenshot of **Mitos** WSE.

When the user interacts with the clustering facet we do not apply the re-clustering process (i.e. steps (1) and (2) of the on-demand algorithm). This behavior is more intuitive, since it preserves the clustering hierarchy while the user interacts with the clustering facet (and does not frustrate the user with unexpected results). In case the user is not satisfied by the available cluster labels for the top- C objects of the answer, he can enforce the execution of the

Faceted taxonomies with on-demand clustering results

Search

Advanced Search

Results per page 10

clusteredDocs(1): #8303870 Results 1 - 10 from 3870 for greece. (2680 ms)

By filetype

By language

By date

By clustering

☒ Rest (3770)
☒ greece (1000)
☒ ??? (11)
☒ yannis (6)
☒ technical reports (11)
☒ symposium (2)
☒ program (3)
☒ presentation (3)
☒ katechakis home page (2)
☒ informatics (7)
☒ ics publications (3)
☒ homepage yannis katechakis (5)
☒ home page (4)
☒ forth (28)
☒ events (4)
☒ ercm (2)
☒ copyright (21)

☐ AND ☒ OR ☐ NOT

Ok

By domain

☒ gr (3870)
☐ OR ☐ NOT

Ok

FORTH-ICS: News - 0.62731844

Greece TELA Conference 2004 17 19 May 2004 Hersonissos Crete **Greece** ... August 25 26 2005 Athens **Greece** <http://www.ics.forth.gr>
<http://www.ics.forth.gr/news/news-prew-2006.html> - 1179398891000 - 37KB [Cached](#) - [Similar pages](#) [\[make as span\]](#)

PrognosisChip - People - 0.6042318

Computer Science **FORTH-ICS Greece** Dimitris Plexousakis Ph D Institute of ... Computer Science **FORTH-ICS Greece** Department of Computer Science University of ...
<http://www.ics.forth.gr/ist/projects/PrognosisChip/people.html> - 1201266607000 - 7KB [Cached](#) - [Similar pages](#) [\[make as span\]](#)

FORTH-ICS: News - 0.6025675

2009 **FORTH Heraklion Crete Greece** <http://www.ics.forth.gr/cv/> ... Orphanoudakis Seminar Room **FORTH Heraklion Crete Greece** 2008 December 2008 TNL
<http://www.ics.forth.gr/news/news-prew.html> - 1241420840000 - 54KB [Cached](#) - [Similar pages](#) [\[make as span\]](#)

FORTH-ICS: Events - 0.5774606

Kolympari Chania Crete **Greece** September 25 26 2006 First Cretan Bioinformatics ... **Greece** June 19 2006 Ygeias Protopyon The 6th seminar was successfully
<http://www.ics.forth.gr/health/events.html> - 1159799354000 - 21KB [Cached](#) - [Similar pages](#) [\[make as span\]](#)

FORTH-ICS: Events - 0.5774606

Kolympari Chania Crete **Greece** September 25 26 2006 First Cretan Bioinformatics ... **Greece** June 19 2006 Ygeias Protopyon The 6th seminar was successfully
<http://www.ics.forth.gr/bioinformatics.html> - 1159799354000 - 21KB [Cached](#) - [Similar pages](#) [\[make as span\]](#)

FORTH-ICS: Events - 0.5774606

Kolympari Chania Crete **Greece** September 25 26 2006 First Cretan Bioinformatics ... **Greece** June 19 2006 Ygeias Protopyon The 6th seminar was successfully
<http://www.ics.forth.gr/health/events.html> - 1159799354000 - 21KB [Cached](#) - [Similar pages](#) [\[make as span\]](#)

4th Hellenic Data Management Symposium - HDMS 2005 - 0.57552415

of Ioannina **Greece** Dimitris Plexousakis University of Crete and **FORTH Greece** ... **Greece** Peter Triantafyllou University of Patras **Greece** Vassilis Tsotras Univ of ...
http://www.ics.forth.gr/hdms05/mell_en.html - 1141121915000 - 15KB [Cached](#) - [Similar pages](#) [\[make as span\]](#)

Figure 4.17: Screenshot of Mitos WSE

clustering algorithm for the next top- C by pressing the *REST* zoom-in point as it has already been mentioned (which keeps pointers to $K - C$ objects).

4.2.2.4 Experimental Results

In this experiment we measured the overall cost, for cluster generation (snippet generation and clustering algorithm execution times) and dynamic taxonomies (to compute the zoom points and to load the new clustering labels to the corresponding facet). Moreover, we compare the non-incremental with the incremental algorithm, which preserves the initial suffix tree and the elimination of old objects is done using the Scan-approach. The scenario includes: (a) the execution of the query *crete* which returns 4067 results, (b) the expansion of the *gr* zoom point of the *By domain* facet and the selection of the *uoc.gr* (1277) zoom-in point from the hierarchy revealed from the expansion, and (c) the selection of the *text/html* (807) zoom-in point of the *By filetype* facet. Let c_a, c_b and c_c be snippets of the top- C elements in the steps (a), (b) and (c) respectively. Figure 4.18 shows the facet terms after steps (a), (b) and (c), as they are displayed in the left bar of the WSE GUI ². We set $K = 10^4$ (i.e. the whole answer set is loaded) and repeated the above steps for the following values of C : 100, 200, ..., 500. We do not measure the cost of the query evaluation time. In all experiments the displayed zoom points are accompanied by count information.

	Step (a)				Step (b)				Step (c)			
	Total	FDT	Clust.	Snip.	Total	FDT	Clust.	Snip.	Total	FDT	Clust.	Snip.
top-100	$ c_a = 100$				$ c_a \cap c_b = 85$, overlap=85%				$ c_b \cap c_c = 22$, overlap=22%			
Non-Incr.	1.72	0.62	0.06	1.04	1.07	0.23	0.03	0.82	0.36	0.21	0.01	0.14
Incr.	1.75	0.62	0.07	1.06	0.53	0.24	0.04	0.25	0.42	0.20	0.09	0.12
top-200	$ c_a = 200$				$ c_a \cap c_b = 174$, overlap=87%				$ c_b \cap c_c = 24$, overlap=12%			
Non-Incr.	2.88	0.49	0.08	2.31	1.95	0.13	0.07	1.76	0.73	0.24	0.13	0.36
Incr.	2.93	0.50	0.11	2.32	0.58	0.13	0.08	0.37	0.66	0.20	0.18	0.28
top-300	$ c_a = 300$				$ c_a \cap c_b = 232$, overlap=77.3%				$ c_b \cap c_c = 78$, overlap=26%			
Non-Incr.	3.51	0.42	0.14	2.95	2.60	0.23	0.16	2.21	0.82	0.21	0.17	0.44
Incr.	3.69	0.5	0.22	2.97	0.7	0.12	0.38	0.2	0.85	0.21	0.31	0.33
top-400	$ c_a = 400$				$ c_a \cap c_b = 257$, overlap=64.25%				$ c_b \cap c_c = 170$, overlap=42.5%			
Non-Incr.	3.87	0.43	0.23	3.21	3.12	0.25	0.31	2.56	2.16	0.12	0.24	1.8
Incr.	4.04	0.5	0.34	3.2	1.09	0.24	0.57	0.28	0.84	0.11	0.45	0.28
top-500	$ c_a = 500$				$ c_a \cap c_b = 269$, overlap=53.8%				$ c_b \cap c_c = 268$, overlap=53.6%			
Non-Incr.	4.39	0.46	0.45	3.48	3.14	0.26	0.32	2.56	2.71	0.24	0.32	2.15
Incr.	4.56	0.45	0.63	3.48	1.32	0.14	0.8	0.38	1.08	0.23	0.56	0.29

Table 4.2: Top- C Integration Timings for non-Incremental and Incremental Algorithms (in seconds)

Table 4.2 shows the intersection of A_c and A'_c for steps (a), (b) and (c) and the execution

²The screenshots are from the previous version of *Mitos* GUI.

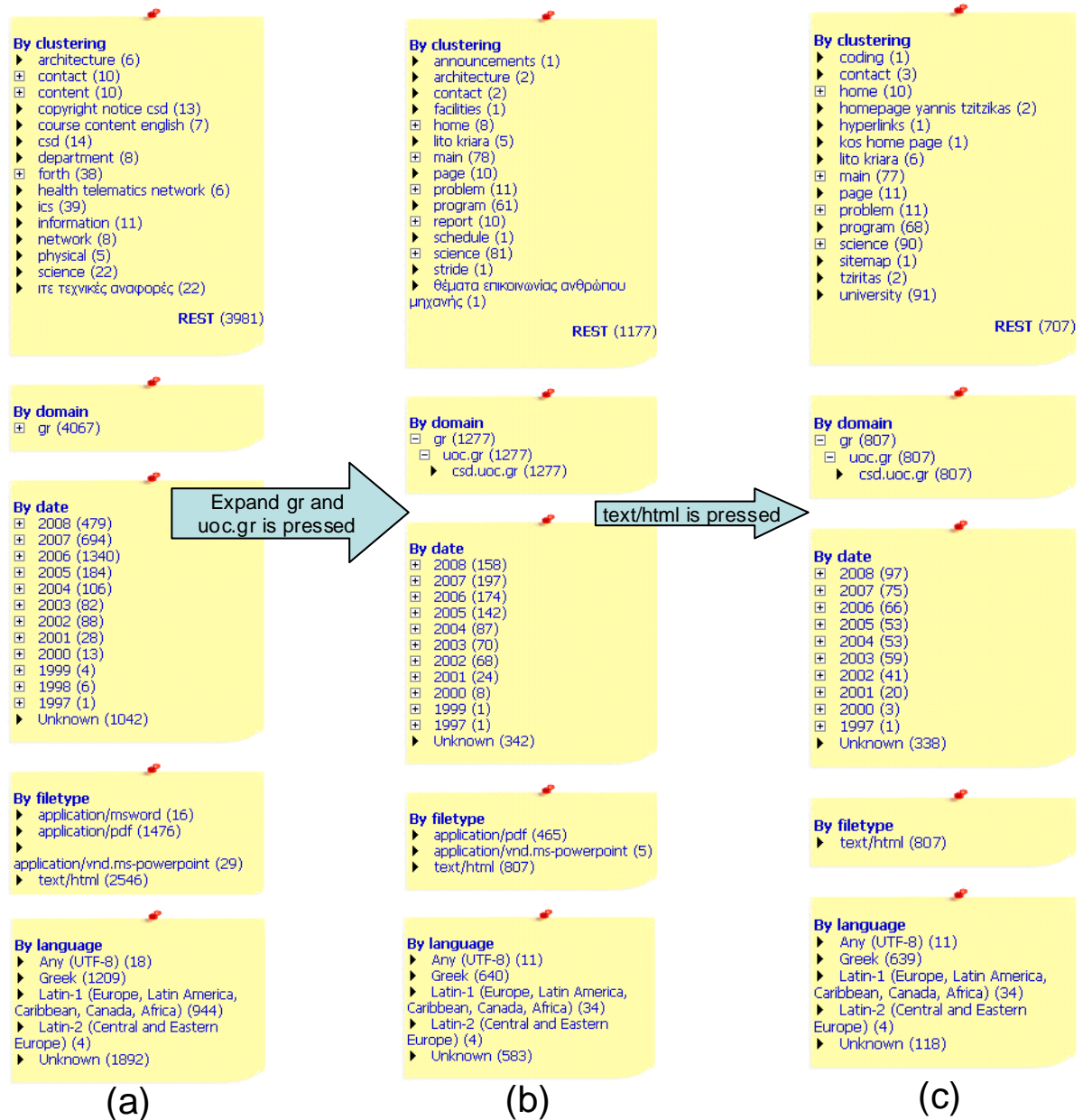


Figure 4.18: Steps (a)-(c) of running scenario

times that correspond to the integration of **flexplorer** and results clustering using the non-incremental and an incremental approach, for the *top - C* elements. Specifically, FDT times include zoom points calculation timings (for each step) and loading to **flexplorer** the entire MFT (step A) and removing and loading the new clustering facet for (steps B and C). It is evident that for top-100 and top-200 values, the results are presented to the user pretty fast (especially for steps B and C), making the proposed on demand clustering method suitable as an online task. Moreover, we can see that there is a linear correlation between time cost and the top-*C* value. Finally, calculating and loading clusters for the top-500 documents, costs less than 4.5 seconds making even big top-*C* a feasible configuration.

4.2.2.5 Evaluation of Usability

We conducted a user evaluation in order to compare the usefulness and usability of three interfaces: (A) one offering FDT but without the results clustering facet, (B) one offering results clustering only, and (C) the proposed interface that combines both.

To this end we specified 4 information needs (or tasks) shown in Table 4.3. For the first three (T1-T3) we specified three variations of each, one for each interface (the subscripts indicate the interface that should be used in each variation). We decided to follow this policy because if we had the same task for each interface then the order by which participants used the interfaces would bias the results (the users would remember the title of the pages that have the sought information). However for the fourth task (*T4*) we did not provide a distinct variation for each interface but the participant had to return the number of the results they found using each interface in a specific interval of time.

For each information need each participant had to fill a form like the one shown at Table 4.4 where apart from the sought information (i.e. the requested URLs), the participant had to rank the three interfaces according to preference by assigning each interface a distinct number from {1,2,3} (1 for the most preferred), and to express the degree of his satisfaction from each interface by using a value from the set {high, medium, low}. The latter question is not comparative in the sense that a user could assign "high" or "low" to all interfaces.

13 users participated in the evaluation with ages ranging from 20 to 30, 61.5% males and 38.5% females. We distinguished the participants into two groups: the *advanced* and the *regular*

Table 4.3: User Evaluation Tasks

Id	Information need/task description
(T1 _A)	Find at least two papers of the head of the Information Systems Laboratory of FORTH-ICS that were published on 2007 and concern Semantic Web.
(T1 _B)	Find at least two papers of the chairman of the Computer Science Department of University of Crete that were published on 2006 and concern e-learning.
(T1 _C)	Find at least two papers of the Director of the ICS-FORTH that were published on 1997 and concern user interfaces.
(T2 _A)	Find the personal web pages of at least two local collaborators of Dimitris Plexousakis.
(T2 _B)	Find the personal web pages of at least two local collaborators of Grigoris Antoniou.
(T2 _C)	Find the personal web pages of at least two local collaborators of Vasilis Christophides.
(T3 _A)	Find two presentations about Wireless Networks in .ppt that are published at the FORTH domain (forth.gr).
(T3 _B)	Find two presentations about Web Services in .ppt that are published at the FORTH domain (forth.gr).
(T3 _C)	Find two presentations about Computer Vision in .ppt that are published at the FORTH domain (forth.gr).
(T4)	Find in 2 minutes all (or the most) persons of CSD who include into their home pages information about their music preferences.

Information Need: (n4)	Response:	
Interface	User Satisfaction (Low/Medium/High)	Preference (1/2/3)
(A) FDT		
Information Need: (n4)	Response:	
Interface	User Satisfaction (Low/Medium/High)	Preference (1/2/3)
(B) Clustering		
Information Need: (n4)	Response:	
Interface	User Satisfaction (Low/Medium/High)	Preference (1/2/3)
Both (A) FDT and (B) Clustering		

Table 4.4: User Evaluation Form

users. The *advanced* users had prior experience in using clustering and multidimensional browsing services, while the *regular* ones had not. For this reason, and before starting the evaluation, we gave to each regular user a brief tutorial on using these services through examples over the *Mitos* WSE. The tutorial was personal (for each participant individually) and lasted around 5 minutes.

Table 4.5 shows the aggregated results of the evaluation for all participants and all tasks per interface. For instance, we can observe that 50% of the *advanced* users had *medium* satisfaction for the *Clustering*. Table 4.7 shows the results of the evaluation for all participants per task. For instance, we can observe that all users had *high* satisfaction for the *FDT* in Task 3.

	Interface	User Satisfaction			Preference			Completeness of Task (n4)
		Low	Medium	High	1	2	3	
Advanced Users	(A) FDT	25%	33.3	41.6	16.6	41.6	41.6	72.2
	(B) Clustering	33.3	50	16.6	25	16.6	58.3	55.5
	Both (A) and (B)	16.6	25	58.3	58.3	41.6	0	77.7
Regular Users	(A) FDT	5	40	55	42.5	25	30	46.6
	(B) Clustering	37.5	50	12.5	20	15	65	38.3
	Both (A) and (B)	12.5	37.5	50	35	57.5	5	45

Table 4.5: User Satisfaction, Preference and Completeness percentage results per Interface

Completeness. The last column of Table 4.5 shows the average percentage of the correct URLs that users found in task *T4* in each user interface, out of the total number of correct URLs in our testbed (6 correct URLs). We observe that with (C) (combination of the *Clustering* with the *FDT*) *advanced* users achieved the highest degree of completeness i.e. 77,7%. *Regular* users achieved the highest degree of completeness using (A) i.e. 46.6% while (C) is quite close, i.e. 45%.

	Queries			Zoom-in Points Clicks								
				By Clustering	By domain		By filetype		By date		By encoding	
	(A)	(B)	Both	Both	(A)	Both	(A)	Both	(A)	Both	(A)	Both
Avg. Advanced	5.6	12.6	5.33	6	1.6	1.3	0.3	0.3	1	1	0	0
Avg. Regular	11.9	13.2	10.9	2.7	1.8	0.8	1	1.1	2	1.7	0.1	0.6

Table 4.6: Number of User Queries and Clicks (as recorded in the log)

Log Data Analysis. We logged and counted the number of queries and clicks (on zoom points) the users made. Specifically for each user we counted: (a) the number of queries submitted and (b) the number of clicks on zoom-points (by facet). Table 4.6 shows the average number of queries and zoom-points that a user from each group made in each interface for all tasks. For example, a *regular* user submitted in average 13.2 queries in *Clustering* interface during the evaluation. At first we observe that both groups submitted the least number of queries when

using (C) interface, which is an evidence that the combination of interfaces makes information seeking less laborious. The difference is significant for the advanced users as they made more than 50% less queries in the *Faceted Taxonomies* and in the combination than the *Clustering*. Regarding clustering zoom points, we can see that a regular user pressed in average only 2.7 zoom-points of the facet *By Clustering*, while an advanced user pressed 6 (we will attempt to explain this difference later on). Notice that 6 clicks on clustering zoom points (that advanced users made) are more than the sum of the clicks on the points of the rest facets.

User Satisfaction For advanced users, (C) seems to be the most preferred choice as the 58,3% of the users rate it first. On the other hand, for *regular* users the most preferred interface seems to be (A) (*FDT*) as 42,5% of them rate it first. In that group the difference in satisfaction between (A) and (C) is small, 55% for the first one and 50% for the second one.

Both groups consider (B)(*Clustering*) as the least preferred interface: 58,3% of the *advanced* users and 65% of the *regular*. This is probably because users are either not familiar with clustering services (as there are only a few meta search engines - and not well known - that offer this feature), or because they do not yet trust such services (as they do not use them regularly).

If we look at the table with the detailed results (Table 4.7) we observe that the advanced users were more satisfied from *Clustering* than from *FDT* for the first two tasks, while in the last two tasks the opposite was happened. The former can not be explained (maybe users were unsatisfied for Task1 with the date facet, since it uses the modification time), but for the latter it is obvious that facets filetype and domain were very helpful for the tasks at hand. *Regular* users were not satisfied from *Clustering* in none task.

Overall, we can conclude that the combination of *FDT* with *Clustering* is expected to help mainly users who are familiar with the functionality of each interface (especially with clustering), and for such users this interface will probably be the most preferred. Users who are not very familiar with these technologies are more satisfied with *FDT* (probably because they fully understand it immediately) than with the combination of both or with *Clustering* alone. This is quite expected as users who have not used real-time clustering probably neither can understand it immediately (e.g. they may wonder whether the clusters contain overlapping or disjoint sets of pages), nor have experience on using such services so they do not trust them. However we have to remark that the tutorial was very brief, and it is possible that a more detailed and comprehensive tutorial (e.g. a hands on training session of 30 minutes) could turn the results of the regular users to converge to those of the advanced ones.

		Interface	User Satisfaction			Preference		
			Low	Medium	High	1	2	3
Advanced	Task1	(A) FDT	66.6	33.3	-	33.3	-	66.6
		(B) Clustering	33.3	33.3	33.3	33.3	33.3	33.3
		Both (A) and (B)	33.3	33.3	33.3	33.3	66.6	-
	Task2	(A) FDT	-	100	-	-	33.3	66.6
		(B) Clustering	-	66.6	33.3	33.3	33.3	33.3
		Both (A) and (B)	-	66.6	33.3	66.6	33.3	-
	Task3	(A) FDT	-	-	100	33.3	66.6	-
		(B) Clustering	66.6	33.3	-	-	-	100
		Both (A) and (B)	-	-	100	66.6	33.3	-
	Task4	(A) FDT	33.3	-	66.6	-	66.6	33.3
		(B) Clustering	33.3	66.6	-	33.3	-	66.6
		Both (A) and (B)	33.3	-	66.6	66.6	33.3	-
Regular	Task1	(A) FDT	10	60	30	20	40	40
		(B) Clustering	20	70	10	20	40	40
		Both (A) and (B)	10	30	60	60	20	20
	Task2	(A) FDT	-	50	50	40	20	40
		(B) Clustering	20	50	30	40	-	60
		Both (A) and (B)	-	60	40	20	80	-
	Task3	(A) FDT	-	-	100	70	20	10
		(B) Clustering	70	30	-	-	10	90
		Both (A) and (B)	30	-	70	30	70	-
	Task4	(A) FDT	10	50	40	40	30	30
		(B) Clustering	40	50	10	20	10	70
		Both (A) and (B)	10	60	30	40	60	-

Table 4.7: User Satisfaction and Preference percentages per Interface (per task)

4.2.3 Exploratory web searching with Entity Mining

We should stress that what we have proposed in section 4.2.2 can be applied also on other kinds of dynamically-mined metadata. With the term *dynamically-mined metadata* we refer to metadata which should be minable (a) from *small quantities or portions* of data, e.g. from the snippets of the top-K part of a query answer, and (b) in real-time. The motivation for focusing on small quantities is that (i) we may not have at our disposal large quantities (e.g. we may have access only to snippets), (ii) it may be computationally expensive to apply these mining tasks on large quantities of data, and (iii) we may want to focus on small qualities for enhancing the quality (specificity) of the mined information.

In the context of web searching, we can say that dynamically-mined metadata refer to metadata which are mined from the snippets of the top elements of the current answer. Examples of such mining tasks, apart from results clustering, include

- *Facet and Taxonomy Mining*

For instance, [12] generates facet hierarchies dynamically from text or text-annotated objects.

- *Entity Mining*

Named entity recognition (also known as entity identification and entity extraction) [28,

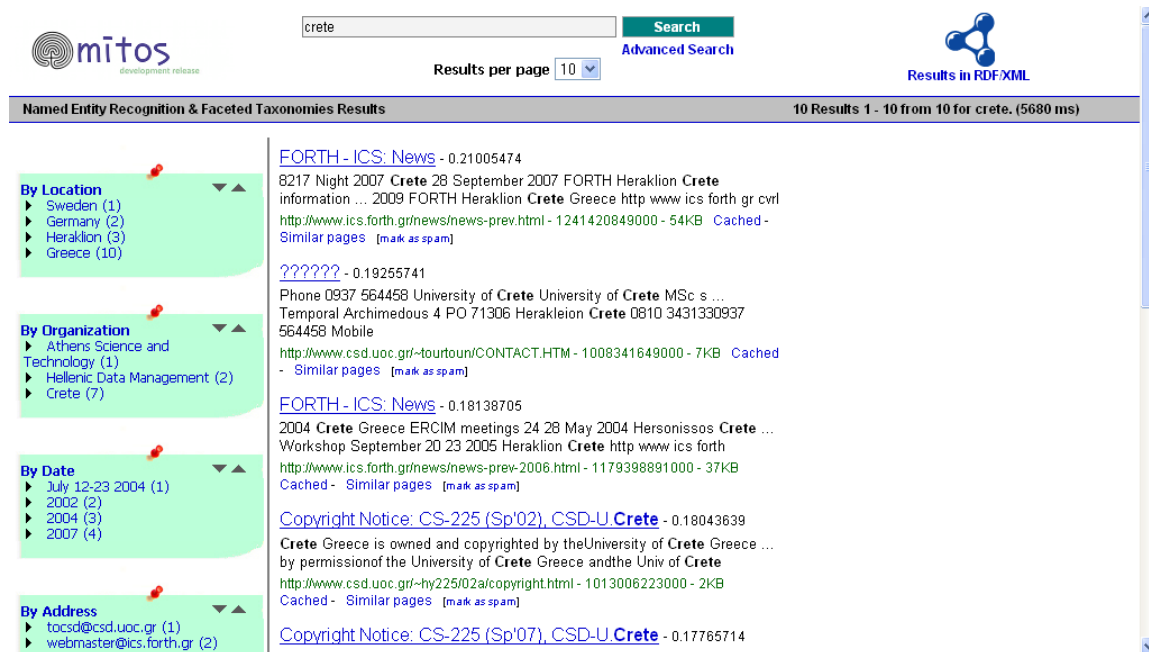


Figure 4.19: Faceted Taxonomies and Entity Mining

30] is a subtask of information extraction that seeks to locate and classify atomic elements in text into predefined categories such as the names of persons, organizations, locations, expressions of times, quantities, monetary values, percentages, etc (e.g. the GNOSIS addon of Firefox).

Nowadays, Mitos supports the coupling of faceted exploration and entity mining using the GATE/ANNIE project [9]. It provides named entity recognition over various format of documents e.g. txt, html, pdf, in several languages and supports various types of entities e.g. location, date. Mitos supports named entity recognition for documents written in English or Greek, and for 6 specific types of entities: Location, Date, Organization, Address, Person and Money.

As regards the combination of entity mining and faceted dynamic taxonomies, we follow the same approach as for clustering with the only difference that in this case each entity is a facet. When the user executes a query, the entity mining is executed on the entire text of the $top - k$ documents of the query answer, where usually $k = 100$. If an entity is recognized in a specific document, then the document is classified under the particular term of the specific entity facet. Figure 4.19 shows the user interface of the specific implementation on Mitos.

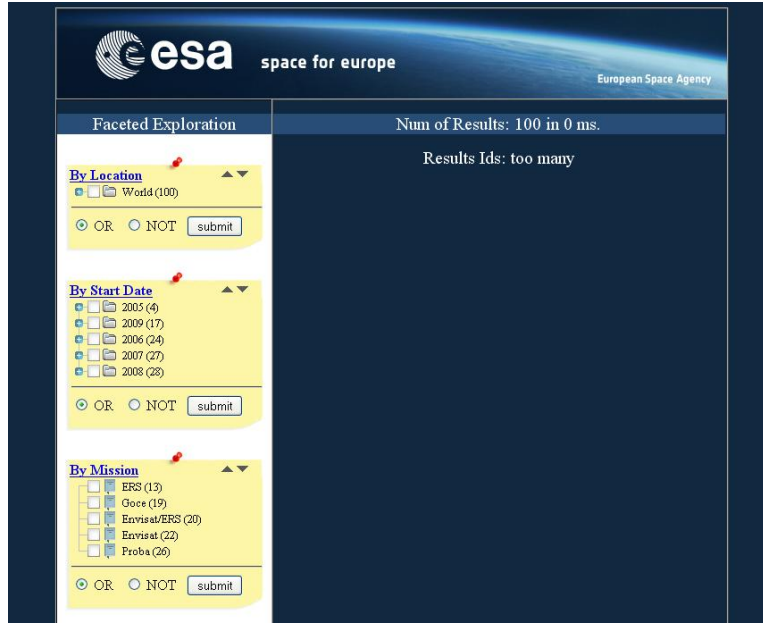


Figure 4.20: ESA-USNG User Interface

4.3 EO User Service Next Generation Project (EO USNG)

The User Services Next Generation is an ESA (European Space Agency) project where the user's needs and requirements are the key driver for improving and redefining the way ESA currently provides its data and services. The improvement of technologies, additional sensors and engaging new user communities are key motivators in defining the new service. ESA aims to increase user visibility of its services and wants to be challenged with a new innovative design that improves the systematic flow and widens the scope of its Earth Observation services. In this project we proposed an improved catalogue searching and browsing approach for ESA Products using Dynamic Taxonomies and Faceted Search.

The first version of web-based user interface from the demonstration phase is presented in Figure 4.20.

4.4 Experimental Results on DB-R Architecture

Here we report experimental results of three different experiments with respect to the hierarchically organized facets and the size of the dataset.

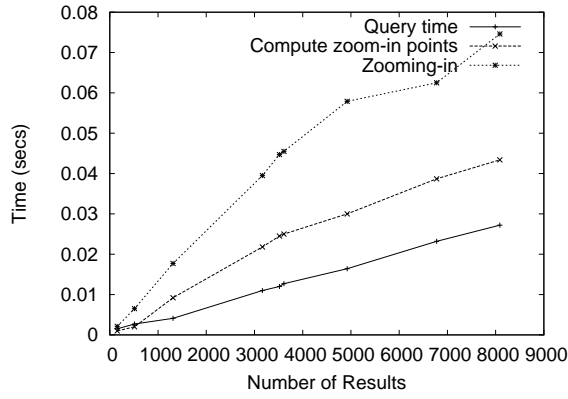


Figure 4.21: Experimental Results on DBMS

4.4.1 (DB-R) With No Hierarchically Organized Values

In this Section we present some experimental results on DB-R approach with no hierarchically organized facets. As the index of *Mitos* is based on a DBMS (specifically PostgreSQL 8.3), we performed experiments over it. The facets presented in Section 4.2.1 actually correspond to columns of the database schema (shown in Table 4.8).

<i>Table</i>	<i>Field</i>	<i>Type</i> (Bytes)
document	id	int4 (4)
	md5	char (16)
	title	varchar (title.length)
	path	varchar (path.length)
	link	varchar (link.length)
	type	varchar (type.length)
	encoding	varchar (encoding.length)
	norm	float (4)
	rank	float (4)

Table 4.8: Partial database schema of *Mitos*.

Figure 4.21 shows the corresponding results (for various result sizes) for computing the zoom-in points for only one facet (whose terms are not hierarchically organized). Notice that the time to compute the contents of the new focus is higher than the time to compute those of the original focus (because the corresponding query is longer). In general, this approach is very fast too.

4.4.2 (DB-R) With Hierarchically Organized Values

Here we investigate the applicability of SQL in case we have hierarchically organized values. However, we should mention that this approach approach is feasible only if we a-priori know the

depth of the taxonomies involved (or if we adopt recursive SQL).

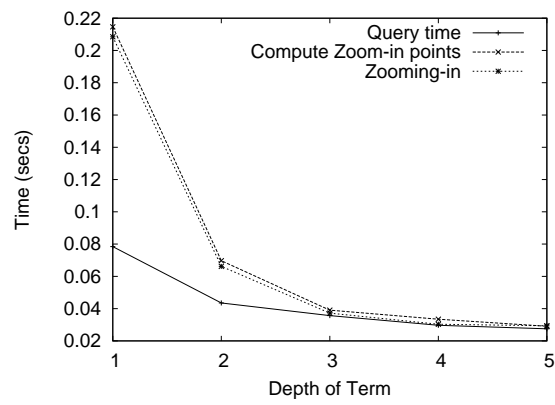
We created a synthetic dataset whose schema is shown in Table 4.9. Table **subjectHierarchy** forms a balanced and complete tree with degree 5 and depth 5. Each paper is associated with one randomly selected subject term (that is a leaf) and with 1 to 4 randomly selected authors. All fields of the tables have been indexed with B-Trees and the size of the database is 30.1 MB (the indexes occupy 17.2 MB). In order to run the experiments we have installed PostgreSQL 8.3 (with shared buffers parameter set to 1 GB) on a Pentium IV machine with 3GHz CPU and 1 GB RAM.

<i>Table</i>	<i>Field</i>	<i>Num.ofTuples</i>
paper	<u>pid</u> title year venue	10^5
author	<u>pid</u> <u>authorName</u>	4×10^4
paperAuthors	<u>pid</u> <u>authorId</u>	2.2×10^5
subjectHierarchy	<u>stId</u> name parentID	3906
paperSubjects	<u>stId</u> <u>pid</u>	10^5

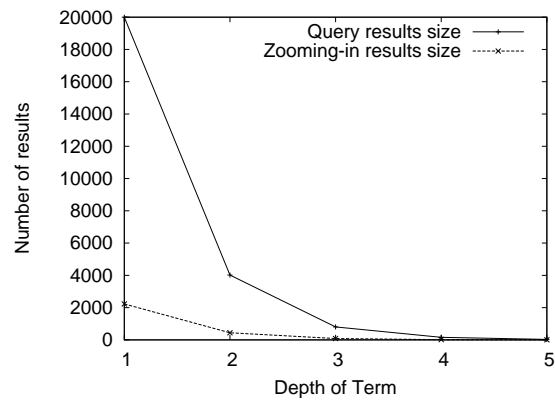
Table 4.9: Database schema of small synthetic dataset

Figure 4.22.(a) shows: (t_a) the time for computing the answer of a query comprising one subject term from various term depths, (t_b) the time to compute the zoom-in points with respect to the **venue** attribute, (t_c) the time to compute the content of the new focus (we have selected one zoom-in point from **venue** facet). Furthermore, the cost t_a is included in both t_b and t_c , since we re-compute the results. The reported times are the average of 20 different runs of 5 randomly selected subject terms for each depth. Figure 4.22.(b) shows the corresponding average result sizes.

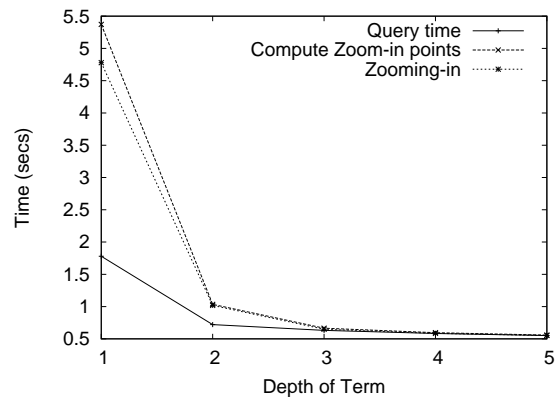
We conclude that the query times increase, compared to the query times in Figure 4.21, for the same number of results. This was an expected result, since to support hierarchically organized values using the DBMS, more complicated queries had to be issued.



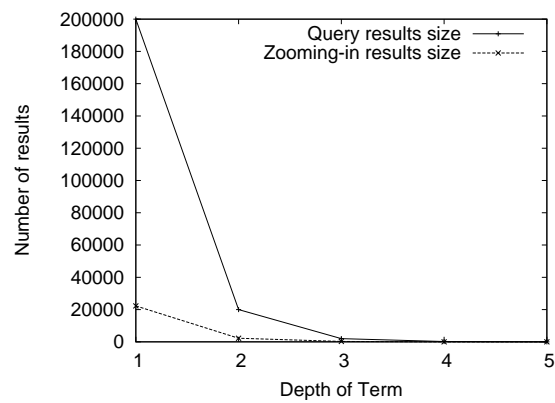
(a)



(b)



(c)



(d)

90
Figure 4.22: Experimental results on synthetic databases.

4.4.3 (DB-R) With Hierarchically Organized Values (Bigger Data Set)

Here we report the results **for larger data sets**, i.e. for databases that do not fit in main memory. We used the database that described in Table 4.10.

<i>Table</i>	<i>Field</i>	<i>Num.ofTuples</i>
paper	<u>pid</u> title year venue	2×10^6
author	<u>pid</u> <u>authorName</u>	5×10^5
paperAuthors	<u>pid</u> <u>authorId</u>	5×10^6
subjectHierarchy	<u>stId</u> name parentId	111.111
paperSubjects	<u>stId</u> <u>pid</u>	2×10^6
subjectHierarchy2	<u>stId</u> name parentId	111.111
paperSubjects2	<u>stId</u> <u>pid</u>	2×10^6

Table 4.10: Database schema of large synthetic dataset.

Tables **subjectHierarchy** and **subjectHierarchy2** form a balanced and complete tree with degree 10 and depth 5. Again, each paper is associated with one randomly selected leaf subject term from each of the two hierarchies and with 1 to 4 randomly selected authors. All fields of the tables have been indexed with B-Tree access method and the size of the database is 1.24 GB (the indexes occupy 718 MB). The experiments run on the same machine as above but with the shared buffers parameter of postgresSQL set to 1 GB. Figure 4.22.(c) shows the times of the same experiment as Figure 4.22.(a) in a larger dataset. The reported times are the average times of 40 different runs for 10 randomly selected subject terms for each depth. Figure 4.22.(d) shows the corresponding average result sizes. The aforementioned times were gathered using Java, meaning that the overhead of the JDBC driver is also included. This overhead also exists in the **Mitos** engine, since it is also written in Java.

Summarizing, when the volume of data increases and performance of the DBMS approach degrades. **Note that 5 seconds is not acceptable for building on-line applications.**

Chapter 5

Extensions For Scalability

Although, faceted and dynamic taxonomies are increasingly used nowadays in a plethora of applications, current methods (which are object-based) are applicable for relative small collections of objects (comprising thousands of objects), and do not fully exploit the characteristics of the information thinning process for reducing the computational costs. The provision of such services for larger collections (e.g. of the magnitude of 10^9) is a challenging vision. In this chapter, we elaborate on techniques that could be used for larger collections, by reducing the associated computational costs and the storage overhead. In more detail, Section 5.1 introduces a scenario that highlights the desired functionality of a FSE over Terra-sized collections. Section 5.2 proposes an effective and efficient interaction scheme for exploring large collections. Section 5.3 evaluates our proposal analytically and experimentally, and discusses the experimental results.

5.1 A Global-scale Exploration Scenario

Suppose that we want to offer FDT exploration services over the set of all web pages. According to the Netcraft¹ the Web (until April 2009) contains 232 million web sites, while as of June 2008, the indexed web contains at least 63 billion pages ².

Suppose that we have a global web search engine which provides faceted exploration functionalities. Let us assume that the faceted taxonomy contains five facets. The facets and their size are described below:

¹http://news.netcraft.com/archives/web_server_survey.html

²<http://www.worldwidewebsize.com/>

- **By language:** According [18], as of early 2007, there are 6,912 known living human languages. So we will have $\simeq 7 * 10^3$ terms for that facet. Furthermore, ISO 639-2 and ISO 639-5 propose a regional taxonomy of language families³. For example, if someone wants to select the Greek language has to follow the path: Europe/South Europe/ Modern Greek. The depth of this path depends on the depth of family hierarchy. The average depth is 3.
- **By TLD (top level domain):** According IANA⁴ the number of top level domains is $271 \simeq 3 * 10^2$. We have four categories of top level domains: gTLD, ccTLD, IDNA TLD and infrastructure. The ccTLD (country code TLD) which includes the 88% of the top level domains can also be categorized by continent. So the average depth of this facet is 2.
- **By date:** In this facet we will have the dates from 1992 to 2012. So, we will have: $(21 * 366) + (21 * 12) + 21 = 7959 \simeq 8 * 10^3$ terms. For selecting a specific date, the user has to follow a path with the following format: year/month/day. Subsequently, the depth of this hierarchy is 3.
- **By filetype:** According IANA, there are 9 categories of Mime Types which are described below: application (701 mime types), audio (118 mime types), example (0 mime types), image (39 mime types), message (19 mime types), model (14 mime types), multipart (14 mime types), text (50 mime types), video (57 mime types). So, the number of terms is $1021 \simeq 10^3$, while the depth is 1.
- **By clustering:** Let us consider the case that we want to offer exploration with respect to a classification scheme like that of DMoz directory⁵. DMoz contains 590.000 categories which are hierarchical organized and the average depth as we try it was 7. In this scenario, the hierarchy is a balanced and complete tree with depth = 7 and degree = 6. The number of terms of this facet will be $\simeq 3.4 * 10^5$.

Table 5.1 summarizes the above. All possible descriptions of this faceted taxonomy (assuming single classification) is: $|T_1 \times \dots \times T_5| \simeq 5.7 * 10^{18}$. Moreover, the number of the terms of the entire faceted taxonomy ($|\mathcal{T}|$) is $3.5 * 10^5$ terms.

³<http://www.loc.gov/standards/iso639-2/iso639-2ra.html> and <http://www.loc.gov/standards/iso639-5/langhome5.html>

⁴<http://www.iana.org>

⁵<http://www.dmoz.org/>

Facet	Terms	Depth
Language	$7 * 10^3$	3
TLD	$3 * 10^2$	2
Date	$8 * 10^3$	3
File Type	10^3	1
Clustering	$3.4 * 10^5$	7
SUM.	$3.5 * 10^5$	-
Avg.	$0.7 * 10^5$	3.2

Table 5.1: Global Web Scenario

According to section 3.2.1, if we have mandatory single classification, as in our scenario, then $C_M = k$ where C_M is the average number of terms that are (directly) assigned to an object $o \in Obj$ and k the number of facets. So in our case, $C_M = 5$. If we assume that $|Obj| = 10^{10}$, it follows that to store the set $D_I(o) \forall o \in Obj$ requires $10^{10} * 5$ ids. Let us assume that the cost of an id is 4 bytes, then the storage overhead will be $10^{10} * 5 * 4 = 2 * 10^{11}$ bytes = 168 GB. As we presented in section 3.2.2, the overhead of storing $D_{\bar{I}}$ instead of D_I , is $d_{M,avg}$ where $d_{M,avg}$ is the average depth of terms that are directly used in object descriptions in materialized faceted taxonomy M . In our case $d_{avg} = 3.2$. From the above description of each facet, we can infer that the majority of objects are classified under the leafs of each hierarchy. So, we suppose that $d_{M,avg} = 3$. Consequently, to store $D_{\bar{I}}(o) \forall o \in Obj$ would require $168 \text{ GB} * 3 \simeq 0.5 \text{ TB}$.

As the space requirements for storing this information is high, the realtime computation of the exact zoom-in points and counts in a web application using ordinary hardware will be unacceptably high.

5.2 Interaction Scheme for Large Collections

There are several works which discuss the challenges of providing faceted exploration services on global web search engines. [50] discusses the challenges of providing such services on a large corpus of documents with many facets. Additionally, [48] discusses the limitation of a search engine to quickly compute (or estimate) the facet values for every result that matches a particular query. However, all performance measurements that have been reported in related works (e.g. in [59, 43, 5]) are over collections consisting of millions of objects, and to the best of our knowledge there is not any system of work that attempts to scale such services for billions or more objects.

For a terra-sized collection, the computation of zoom-in points in real time, is roughly impossible for the methods presented in section 3.3 using a single PC and assuming the current technology. To this end, we propose a variation of the interaction scheme of FDT that provides count information only if the focus size is under a certain threshold. For instance, assume that the size of the collection is 10^9 . The count information for the corresponding zoom-points, are not very useful. A rough approximation of the count information (an upper and lower bound) for the computation of the right zoom points would be enough. For this reason, the designer should be able to define a threshold, e.g. $thres = 10^6$, and when the user after a number of clicks reaches a focus whose the upper bound of the count information is below $thres$ then zoom-in points with exact count information will be computed.

In more detail, this variation proposes two different interaction approaches: (i) if the focus size is over a certain threshold we will only compute the zoom-in points for each facet and their counts approximately, and (ii) if the focus size is under the threshold we will follow the classical object-based methods (remember chapter 3). In the first interaction approach we will follow the *CTCA-based* approach for providing the specific services, while in the second interaction approach the *TLOI-based*. Below we discuss each approach separately.

According to the *CTCA-based* approach, we compute the zoom-in points but we provide approximated count information and we do not compute the focus answer. To compute zoom-in points with no count information, we need a kind of preprocessing that yields some data that can be exploited for speeding up the computation. However, these data should have low storage space requirements. For this problem we adopt the Compound Term Composition Algebra (CTCA) [55]. CTCA is fully *intensional*, in contrast to dynamic taxonomies which are both *intensional* (due to the existence of hierarchies and their semantics) and *extensional* (as they discard queries with empty extension). The adoption of CTCA allows computing the zoom-in points without having to perform any operation on the object-base. Instead what we have to do is to mine offline a CTCA expression that specifies all conjunctions of terms whose answer is non empty and at run-time to reason over the mined CTCA expression. In this way we bypass the overhead of the object-based approaches. The required CTCA expression is expected to have low storage space requirements due to the compressing potential of CTCA (for more see [51]).

On the other hand, in case of *TLOI-based* approach, we should compute the zoom-in points, provide the exact count information and compute the answer of the focus. Such information cannot be derived from *CTCA-based* approach. This is an object-based method but it supports

efficient storage indices which reduce the computational costs.

We have to mention that the proposed interaction scheme supports the FDT exploration of an entire collection and not a party of it e.g. FDT exploration on a subset $A \subseteq Obj$ where A has been provided by an external access method like an answer to a query in a WSE. In other words, in this case we do not support the $feed(A)$ operation presented in section 3.1.1.3, because all computations are based in pre-computed indices which do not support for the moment this functionality. This is an issue for further research.

Below we describe each approach analytically. In more detail, section 5.2.1 elaborates on the *CTCA-based* approach, while section 5.2.2 on the *TLOI-based* approach. Moreover, in section 5.2.1.5 we propose a formula for estimating approximately the count information of a focus in order to identify when we should apply the *TLOI-based* approach and when the *CTCA-based*.

5.2.1 CTCA-based Approach

This section describes how CTCA is exploited. The subsequent sections give more details for various parts.

5.2.1.1 V_i & EV_i Computation

Let $M = (\mathcal{F}, I)$, be the materialized ontology. Let $V(M) = \{s \subseteq \mathcal{T} \mid \bar{I}(s) \neq \emptyset\}$ i.e. the set of all compound terms that have not empty extension. By using the approach described in [53] we can produce an expression e of CTCA, such that $S_e = V(M)$, i.e. the terms that are valid according to CTCA expression e , are those that are extensionally not empty in M . This means that we can store the expression e and whenever we want to see if $s \in S_e$ we employ the algorithm described in [55].

Let us now describe how the above can be exploited in the interaction paradigm of FDT. Let st denotes the current state of the interaction with respect to the interaction states presented in section 3.1. As we have already noted, in this case we can only execute $click(t)$ operations. Furthermore, let us assume that we have a function $isValid(e, s)$ which returns *True*, if s is a valid compound term according to e (i.e if $s \in S_e$) and *False*, otherwise.

In case of Simple Visualization Mode (*SVM*):

$$V_i(st) = \{t \in N(st.ctx_i) \cup Inc_i(st.ctx_i) \mid isValid(e, st.ctx \cup \{t\}) = True\} \cup B^*(st.ctx_i)$$

In case of Extended Visualization Mode (*EVM*):

$$EV_i(st) = \{t \in \mathcal{T}_i \mid isValid(e, st.ctx \cup \{t\}) = True\}$$

5.2.1.2 A short introduction to CTCA

CTCA is used for specifying the set of compound terms over a given faceted taxonomy that are valid (i.e. meaningful) in the application domain. From a "logical" point of view, we could say that CTCA is an algebra for specifying the "satisfiable" conjunctions of terms.

As we described above, if e is an expression, S_e denotes the outcome of this expression and is called the compound terminology of e . The initial operands, thus the building blocks of the algebra, are the basic compound terminologies, which are the facet terminologies with the only difference that each term (for reasons of notational simplicity) is viewed a singleton. Specifically, the basic compound terminology of a terminology T_i is defined as: $T_i = \{\{t\} \mid t \in T_i\} \cup \{\emptyset\}$.

CTCA provides four basic algebraic operators: *plus-product* (\oplus_P), *minus-product* (\ominus_N), *plus-self-product* (\oplus_P^*) and *minus-self-product* (\ominus_N^*). The definition of each operation of CTCA is summarized in Table 5.2. They are all operations over $P(\mathcal{T})$. Each of these four operations has an extra parameter denoted by P or N , respectively. The set P is a set of compound terms that we certainly want to appear in the result of the operation, i.e. they are valid. From these more valid terms are inferred. On the other hand, the set N is a set of compound terms that we certainly do not want to appear in the result of the operation, i.e. they are invalid. From these more invalid terms are inferred.

Product operation is an auxiliary operation that results in an "unqualified" compound terminology whose compound terms are all possible unions of compound terms from its arguments. Here we have to mention that a *product* operation is equal with a *minus-product* operation if N is empty. An example is shown in Figure 5.1. On the other hand, a *self-product* operation gives all possible compound terms of one facet. *Plus-self-product* operation results in a compound terminology consisting of the compound terms of the initial basic compound terminology, plus all compound terms which are broader than an element of P , while *minus-self-product* the opposite. For example, the result of the operation $\oplus_P^*(BySports)$, where $P = \{\{SeaSki, Windsurfing\}, \{SnowSki, Snowboard\}\}$ is shown in Figure 5.2.

An expression e over \mathcal{F} is defined according to the following grammar ($i = 1, \dots, k$) in BNF where k is the number of facets:

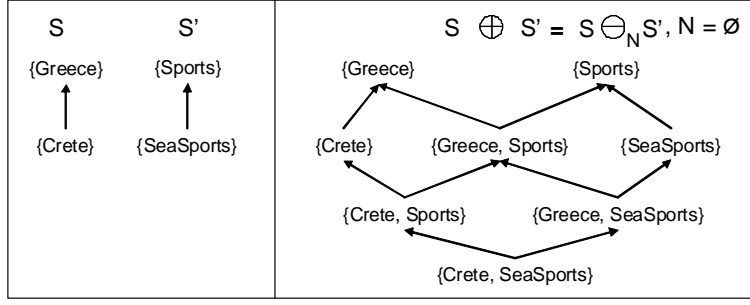


Figure 5.1: Product and minus-product operation example

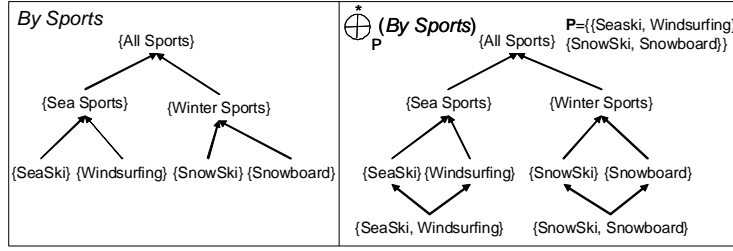


Figure 5.2: Self-plus-product operation example

$$e ::= \oplus_P(e, \dots, e) \mid \ominus_N(e, \dots, e) \mid \oplus_P^* T_i \mid \ominus_N^* T_i \mid T_i$$

Let us assume that we have the materialized faceted taxonomy of Figure 3.3. This materialized faceted taxonomy will be the running example of this chapter. In order to understand the usability of *minus* and *plus* products, an expression e that defines a set of compound terms S_e which are those that have no empty extension is the following: $e = ByLocation \ominus_N BySports$ where

$$N = \{\{Crete, WinterSports\}, \{Olympus, Sea Sports\}, \\ \{Heraklio, Seaski\}, \{Ios, WinterSports\}, \\ \{Ios, SeaSki\}, \{Lasithi, Windsurfing\}\}$$

There are several expressions that can be used to specify the same set of compound terms. For example we can use an expression e' that has a *plus-product*: $e' = ByLocation \oplus_P BySports$ where

$$P = \{\{Olympus, Snowboard\}, \{Olympus, SnowSki\}, \\ \{Heraklio, Windsurfing\}, \{Ios, Windsurfing\}, \\ \{Lasithi, Seaski\}\}$$

Operation	e	S_e
<i>product</i>	$S_1 \oplus S_2 \oplus \dots \oplus S_k$	$\{s_1 \cup s_2 \cup \dots \cup s_k \mid s_i \in S_i\}$
plus-product	$\oplus_P(S_1, S_2, \dots, S_k)$	$S_1 \cup S_2 \cup \dots \cup S_k \cup B^+(P)$
minus-product	$\ominus_N(S_1, S_2, \dots, S_k)$	$S_1 \oplus S_2 \oplus \dots \oplus S_k - N^+(N)$
<i>self-product</i>	$\oplus^*(T_i)$	$P(T_i)$
self-plus-product	$\oplus_P^*(T_i)$	$T_i \cup B^+(P)$
self-minus-product	$\ominus_N^*(T_i)$	$\oplus^*(T_i) - N^+(N)$

Table 5.2: Basic notions and notations

As our paradigm is too small and it does not show the power of CTCA, imagine that we add the term *Islands* in $\mathcal{T}_{ByLocation}$ where $Islands \prec Greece$, $Ios \prec Islands$ and $Crete \prec Islands$ and three objects $\{H6, H7, H8\}$ where $D_I(H6) = \{Ios, Seaski\}$, $D_I(H7) = \{Heraklio, Seaski\}$, and $D_I(H8) = \{Lasithi, Windsurfing\}$. Then we can use the expression:

$e'' = ByLocation \ominus_N BySports$ where $N = \{\{Islands, WinterSports\}, \{Olympus, SeaSports\}\}$.

As we saw above, there are several expressions that could be used for defining the same partition. In particular, what we are looking for is a Sperner system [46] of the maximal invalid compound terms in case of N or the minimal valid compound terms in case of P .

One system based on CTCA has already been built [57], while other applications of CTCA are described in [51, 53]. Approaches for mining the expression e have been already proposed and are discussed in Section 5.2.1.4.

5.2.1.3 Compound term validity and CTCA

To check the validity of a compound term s , we can use the algorithm $IsValid(e, s)$ as described in [55] and presented in Alg.1 which returns *True*, if $s \in S_e$ and *False*, otherwise.

Depending on the parse tree of e , i.e on the kind and number of operations (\oplus_P , \ominus_N , \oplus_P^* , \ominus_N^*) that are used in e , the algorithm contains steps of the form:

- (1) if $\exists n \in N$ s.t. $n \succeq s$, then s will be invalid; valid otherwise (appears in lines 14-21 and 29-33).
- (2) if $\exists p \in P$ s.t. $p \preceq s$, then s will be valid; invalid otherwise (appears in lines 7-13 and 22-28).
- (3) if $\nexists t \in \mathcal{T}_i, \forall i = 1, \dots, k$ s.t. $\{t\} \preceq s_i$ ⁶, then s will be invalid; valid otherwise (appears in

⁶ s_i may contain more than one term of facet i

Algorithm 1 $\text{IsValid}(e, \mathcal{F}, s)$

```

1: if ( $s = \emptyset$ ) then
2:   return ( $TRUE$ )
3: if ( $\exists t \in s$  such that  $F(t) \notin F(e),$ ) then
4:   return ( $FALSE$ )
5: if ( $s$  is singleton) then
6:   return ( $TRUE$ )
7: if ( $e$  instanceOf  $\oplus_P(e_1, \dots, e_n)$ ) then
8:   if ( $\exists p \in P$  s.t.  $p \preceq s$ ) then
9:     return ( $TRUE$ )
10:  for ( $i = 1$  to  $n$ ) do
11:    if ( $\text{IsValid}(e_i, \mathcal{F}, s) = TRUE$ ) then
12:      return ( $TRUE$ )
13:  return ( $FALSE$ )
14: else if ( $e$  instanceOf  $\ominus_N(e_1, \dots, e_n)$ ) then
15:   if ( $\exists n \in N$  s.t.  $s \preceq n$ ) then
16:     return ( $FALSE$ )
17:   for ( $i = 1$  to  $n$ ) do
18:      $s_i = \{t \in s \mid F(t) \in F(e_i)\}$ 
19:     if ( $\text{IsValid}(e_i, \mathcal{F}, s_i) = FALSE$ ) then
20:       return ( $FALSE$ )
21:   return ( $TRUE$ )
22: else if ( $e$  instanceOf  $\oplus_P^*(T_i)$ ) then
23:   if ( $\exists p \in P$  s.t.  $p \preceq s$ ) then
24:     return ( $TRUE$ )
25:   if ( $\exists t \in \mathcal{T}_i$  s.t.  $\{t\} \preceq s$ ) then ▷ i.e.  $s \in T_i$ 
26:     return ( $TRUE$ )
27:   else
28:     return ( $FALSE$ )
29: else if ( $e$  instanceOf  $\ominus_N^*(T_i)$ ) then
30:   if ( $\exists n \in N$  s.t.  $s \preceq n$ ) then
31:     return ( $FALSE$ )
32:   else
33:     return ( $TRUE$ )
34: else if ( $e$  instanceOf  $T_i$ ) then
35:   if ( $\exists t \in \mathcal{T}_i$  s.t.  $\{t\} \preceq s$ ) then ▷ i.e.  $s \in T_i$ 
36:     return ( $TRUE$ )
37:   else
38:     return ( $FALSE$ )

```

lines 34-38).

While CTCA approach has low storage overhead as we have only to store the defining algebraic expression e and the compound terms in \mathcal{P}, \mathcal{N} , the complexity of $IsValid(e, s)$ is $O(|\mathcal{T}|^3 * |s| * |\mathcal{P} \cup \mathcal{N}|)$ where \mathcal{P} is the union of the P parameters of all *plus-product* operations and \mathcal{N} is the union of the N parameters of all *minus-product* operations which exist in e [54]. This complexity overhead owed to the transitive closure computations and to the sequential searching of \mathcal{P} and \mathcal{N} parameters, as no index is supported.

In this thesis we propose optimizations to minimize this overhead. In more detail, section 5.2.1.6 proposes the usage of labeling algorithms for avoiding the cost of transitive closure computations, and efficient indices for storing the \mathcal{P}, \mathcal{N} parameters.

5.2.1.4 Mining a CTCA expression

A CTCA expression can be formulated manually or automatically. For instance a designer can use CTCA in order to specify the set V of valid compound terms in a flexible and gradual manner, without having to provide explicitly every element of V (the manual specification of the elements of V would be a formidably laborious task). Note that if we have a materialized faceted taxonomy $M = (\mathcal{F}, I)$, as in our case we could mine the expression e using the approach presented in [53] (this is called expression mining). The cost of mining the expression e according [53] is:

$$O\left(\frac{|\mathcal{T}|^{k+2}}{k^{k-1}}|Obj| + \frac{|\mathcal{T}|^{2k+2}}{k^{2k}} + (k-1)!5^k * Task(spo)\right)$$

where k is the number of facets and $Task(spo)$ is an optimized process where the parse tree of an expression e is enriched with the \mathcal{P}, \mathcal{N} parameters while its complexity is at the magnitude of $O(|\mathcal{T}|^2)$. The time complexity is polynomial with respect to \mathcal{T} and exponential with respect to k but this algorithm will run once (offline) and the most times the k is small (in our example $k = 5$).

5.2.1.5 Approximating Zoom Point Count Information

As we have already described in section 5.2, we need a formula for estimating approximately the size of focus answer e.g. $|\bar{I}(ctx)|$, in order to specify which interaction approach we have to follow. In this section we propose a method for approximating the count information of a focus ctx in constant time.

Suppose that we know the counts of all terms of the faceted taxonomy, where $count(t) = |\bar{I}(t)|$. The *count* of t is the number of objects that are indexed with t or its descendants. In case we want to compute the count of a compound term $s = \{t_1, \dots, t_k\}$ approximately, we need to compute the *upper* and the *lower bound* of $|\bar{I}(s)|$. The bounds are the following:

$$\text{Upper Bound: } UB(s) = \min_{i=1}^k (count(t_i)),$$

$$\text{Lower Bound: } LB(s) = 0$$

Clearly, $UB(s)$ is the minimum count of the term in s since $\bar{I}(s) = \cap_{i=1}^k \bar{I}(t_i)$. For example, if $s = \{Creece, SeaSports\}$ in our running example, then $LB(s) = 0$ and $UB(s) = \min(|\bar{I}(Creece)|, |\bar{I}(SeaSports)|) = \min(5, 3) = 3$

Definition 3 We call a MFT *cartesian* if each object is mandatorily indexed by one and only one term from each facet.

Prop. 1 (Lower Bound)

In a cartesian materialized faceted taxonomy, if $t_1, \dots, t_k \in \times_{i=1}^k T_i$, then

$$LB(|\bar{I}(t_1, \dots, t_k)|) = \max(0, \sum_{i=1}^k |\bar{I}(t_i)| - (k-1)|Obj|) \quad \square$$

For example, suppose we have 3 facets A, B, C with $\mathcal{T}_A = \{a_1, a_2\}$, $\mathcal{T}_B = \{b_1, b_2\}$, $\mathcal{T}_C = \{c_1, c_2\}$ and 4 objects where each object is indexed with exactly one term from each facet. Let us assume that $ctx = \{a_1, b_1, c_1\}$, $count(a_1) = count(b_1) = 4$ and $count(c_1) = 2$. It is obvious that $count(ctx) = 2$ as all objects are described by a_1 and b_1 , but only two of them are described by c_1 . So $LB(ctx) = 4 + 4 + 2 - 2 * 4 = 2$.

5.2.1.6 Optimizations

In this section we present two optimizations for minimizing the computational costs of *isValid* algorithm. In more detail, the optimizations are: (i) *labeled taxonomies & naive CTCA's parameters indexing*, and (ii) *FDT-based Method for storing the CTCA's parameters*. Below we present analytically the optimization methods, while section 5.3 presents experimental results on these optimizations.

Labeled Taxonomies & naive CTCA's parameters indexing

In order to avoid the cost of subsumption checking (remember $O(|\mathcal{T}|^2)$) we can use a *labeling algorithm* that allows deciding subsumption in constant time, like the one proposed by Agrawal et al. [4] which relies on the introduction of a spanning tree to distinguish between tree and

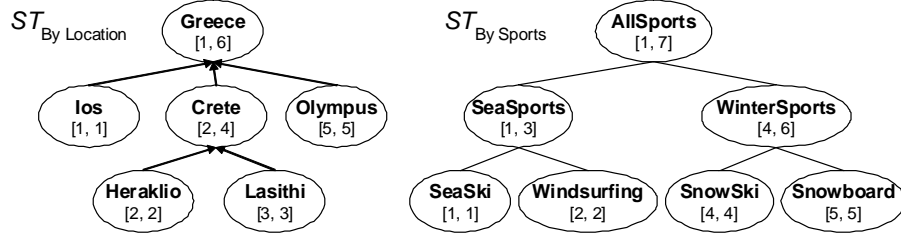


Figure 5.3: Labeling algorithm over the Faceted Taxonomy

non-tree edges. Specifically, they propose a hybrid scheme in which the spanning tree edges fully take advantage of the interval-based labeling, while the non spanning tree edges require a replication of the label of their source node upwards to their target and its ancestors. Then, subsumption checking for spanning tree edges relies purely on interval inclusion test, while for the remaining edges one has to also check whether there is a path in the graph.

More precisely, a node u in the spanning tree ST of the graph is labeled with an interval $[index, postorder]$ where $postorder$ is the number of u in order to reflect its relative position in a postorder traversal of the tree and $index$ is the lowest postorder number among its descendants. Now, for checking the subsumption $u \leq u'$, let the postorder number of u be u_{pn} and the index number be u_i , and for u' be u'_{pn} and u'_i respectively. There exists a direct path from node u to u' iff $u_i \geq u'_i$ and $u'_{pn} > u_{pn}$.

In our approach we will have a spanning tree for each F_i where the nodes are the terms of \mathcal{T}_i . This compression scheme of transitive closure requires $O(|\mathcal{T}|)$ storage. Figure 5.6 presents the labels that will be created for the hierarchies presented in figure 3.3.

Recall that according to Table 2.1, *compound ordering* is defined as $s \preceq s'$ iff $\forall t' \in s' \exists t \in s$ s.t. $t \leq t'$. As we describe above, the cost of checking $t \leq t'$ is $O(1)$. So the overall cost of subsumption checking of two compound terms s, s' in case that all taxonomies are labeled will be $O(\min(|s|, |s'|))$.

Then, a naive method for storing the \mathcal{N} (or \mathcal{P}) can be used. For each $n \in \mathcal{N}$, one interval (w.r.t. the labeling algorithm) for each of its facets (i.e. for each $n_i = n \cap T_i$ and $i = 1, \dots, k$) is stored. We can do the same for the \mathcal{P} parameter.

In case a valid/invalid compound term belongs to a self plus/minus product operation parameter, then the cell for the specific facet will contain a set of intervals while the other cells will be empty. The time complexity for checking whether there exists a compound term that belong to \mathcal{P}, \mathcal{N} which determines whether a focus ctx is valid or not, is $O(|\mathcal{N} \cup \mathcal{P}| * |k|)$ while

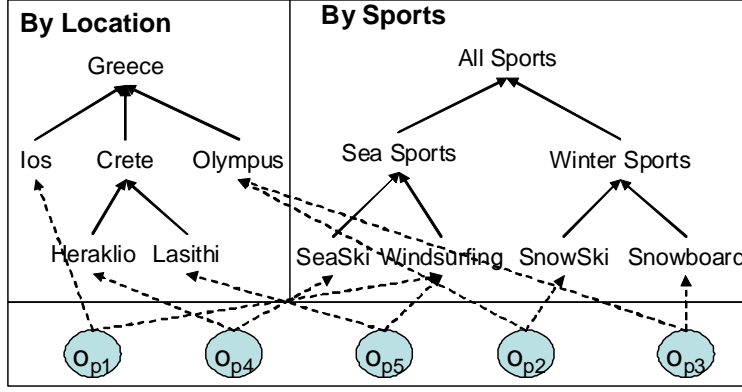


Figure 5.4: Indices for storing \mathcal{P} parameter

the storage overhead is $|\mathcal{N} \cup P| * k$, as we need to check all the parameters and the cost of subsumption checking of ctx is $|k|$.

FDT-based Method for storing the CTCA's parameters

Here we describe an alternative approach for storing $\mathcal{N} \cup P$. **Plus-Product:** The rough idea is the following: for each $p \in P$ we create one artificial object o_p which is classified under the terms that constitute p and this defines an auxiliary interpretation, say Y . Then we reduce the problem of deciding whether an s is valid by checking whether $\bar{Y}(s)$ is non empty. For example, Figure 5.4 shows the materialized taxonomy of the running example with respect to the \mathcal{P} . For $ctx = \{Heraklio, Snowski\}$ we can see that $\bar{Y}(ctx)$ is empty so ctx is invalid.

Minus-Product: Analogously, for each $n \in N$ we create one artificial object o_n which is classified under the terms that constitute n and this defines an auxiliary interpretation, say Y . However from Y we will not define \bar{Y} (as for plus-products), but a new interpretation denoted by \underline{Y} which is defined as $\underline{Y}(t) = \cup\{Y(t') \mid t' \geq t\}$, i.e. it is like propagating the objects downwards in the hierarchy. We reduce the problem of deciding whether an s is valid by checking whether $\underline{Y}(s)$ is non empty. For example, Figure 5.5(a) shows the materialized taxonomy of the running example with respect to the \mathcal{N} , while figure 5.5(b) shows the interpretations of the terms. For $ctx = \{Heraklio, SnowSki\}$ we have $\underline{Y}(Heraklio) = \{o_{n3}, o_{n4}\}$ while $\underline{Y}(SnowSki) = \{o_{n2}, o_{n3}\}$, so $\underline{Y}(ctx) = \{o_{n3}\}$ and ctx is invalid.

The storage overhead and the time complexity of this approach have been already presented in chapter 3.

It is obvious that using the FDT-based method we do not need to label the taxonomies and store the \mathcal{N}, P using the naive method. So we can follow only one of the proposed optimizations.

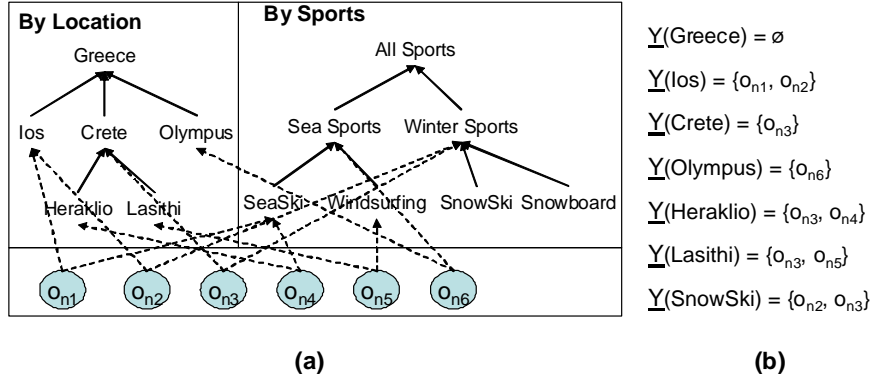


Figure 5.5: Indices for storing \mathcal{N} parameter

The experimental results presented in section 5.3.3 specify which optimization is preferable and under which conditions.

5.2.1.7 Related Work on Labeling Schemes

As we mentioned, a labeling algorithm allows deciding subsumption in constant time. Roughly, three kinds of labeling algorithms have been proposed: *prefix-based*, *interval-based* and *bit-vector-based*. In this section we present the *prefix-based* and *bit-vector-based* schemes, and we discuss why they are not proper for our approach.

A *prefix-based scheme* directly encodes the parent of a node in a tree, as a prefix of its label using for instance a depth-first tree traversal. The subsumption checking in *prefix-based* schemes is performed by comparing two strings (labels) while the storage required for the labels of a tree Tr is $O(|Tr|)$ and the size of the proper node label at each level depends only on the maximum depth of Tr . An interesting property of prefix-based labels is their lexicographic order: the labels of nodes u in a subtree with root v are greater (smaller) than those of its left (right) sibling subtrees. *Dewey Decimal Coding (DDC)* is a labeling scheme which belongs to this category [1]. It is widely used by librarians and further investigated in [8, 15]. Figure 5.6 presents the labels that will be created in our running example according to DDC.

However, the *interval-based* schemes are more efficient than *prefix-based* as subsumption checking is executed by comparing four integers and not Strings. Furthermore, from the storage point of view, the *interval-based* approach is better as we have to store only two integers for each $t \in \mathcal{T}$ than a String in *prefix-based* approach.

In *bit-vector-based* labeling schemes, the label of a node in a tree Tr is represented by a

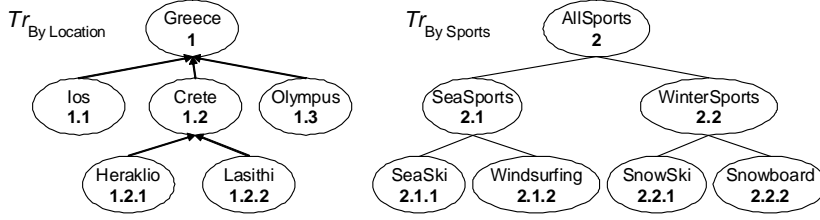


Figure 5.6: DDC labeling algorithm over the Faceted Taxonomy

vector of $|Tr|$ bits, a bit set to "1" at some position uniquely identifies the node in a lattice L and each node inherits the bits identifying its ancestors (or descendants) in a top-down (or bottom-up) encoding. The *bit-vector-based* approach is not beneficial as $|Tr|$ can be very high.

5.2.2 TLOI-based Approach

According to the proposed interaction scheme, we have to follow the *TLOI-based* approach when the focus size is under a certain threshold. In this case we need to compute the zoom-in points, to provide the exact count information and to compute the answer of the focus. It is obvious that we need to follow an object-based method. In this section we present an object-based method named *TLOI-based* (Term Labeling by Objects Ids) which supports efficient storage indices for minimizing the required computational costs and the storage overhead. In more detail, we propose indices for storing the taxonomies and the interpretations of a cartesian materialized faceted taxonomy. Moreover, we present algorithms for computing the zoom-in points.

5.2.2.1 Indices for Storing the Interpretations

In section 3.2, we concluded that to provide the exact count information the *maximal storage* policy is preferable. Furthermore, according to the complexities of zoom-in points computation, in case of very large collections where A is also very big the *Extension Intersection-based* evaluation approach is more efficient. Subsequently, the most efficient and effective storage policy we have to follow is (\bar{I}, \leq^r) , where $\forall t \in \mathcal{T}$ we store the $\bar{I}(t)$ and the $N(t), B(t)$. Moreover, we determined that the usage of indices for storing the \bar{I} and A , like hash-based indices, minimizes the computational costs. However, the storage overhead for storing the $\forall t \in \mathcal{T}$ the $\bar{I}(t)$ is $|Obj| * C_M * d_{M,avg}$. Consequently, we need an efficient index for storing the $\bar{I}(t), \forall t \in \mathcal{T}$ which will reduce both computational costs and storage requirements.

As we described in Section 5.2, when the user (through a sequence of clicks) reaches a focus

whose upper bound object cardinality is below $thres$, then zoom-in points with exact count information should be computed. However, the computation of $\bar{I}(ctx)$ can be very expensive at some occasions, despite the fact that $UB(ctx) \leq thres$, as the computation is based on the intersections between the $\bar{I}(ctx_i)$, $ctx_i \in ctx$. For instance, this can happen in the cases of the form: assume that $thres = 10^6$, and consider a ctx such that $|ctx| = 5$ where four of these terms have $|\bar{I}(\cdot)| = 10^8$, and one has 10^6 . This overhead can be avoided by using hash-based indices for storing the interpretations. However, returning to the object-based approaches, recall that if $|Obj|$ is high then \bar{I} can also be very high incurring a big storage overhead.

To tackle this problem we introduce a novel approach (index), that we call TLOI, which can significantly reduce the required time and storage overhead for *cartesian* and hierarchically organized MFTs⁷. TLOI is constructed as follows: we use the Depth-first search (DFS) algorithm for traversing the hierarchy of a facet and the term-to-object associations⁸, and we give a unique integer (id) to each object o the first time we encounter it. The ids are contiguous, so each $\bar{I}(t)$ is represented as an interval defined by the min and the max object identifiers that belong to $\bar{I}(t)$. We have to mention that this approach can be used only in cartesian MFTs (i.e. when we have single classification), because in case of multiple classification an object will have more than one ids for a facet. We do the above procedure for each facet. At the end, each object will have k ids where k is the number of the facets.

Regarding storage, for each $t \in \mathcal{T}$ we store the interval that corresponds to the labeled objects in $\bar{I}(t)$. Furthermore, we create two indices: i) $\forall o \in Obj$ we store the ids of o for each facet, and ii) for each $id \in [1, |Obj|]$ we store the object which it describes for each facet. Figure 5.7(a) shows the inverted index that is created for our running example while Figure 5.7(b) shows the objects indices.

So, the space required is $2 * |Obj| * k$. According to 3.2.2, the storage overhead if we want to store the $\bar{I}(t), \forall t \in \mathcal{T}$ is $|Obj| * d_{M,avg} * C_M$. In our case $C_M = k$ so from the storage point of view it follows that:

TLOI is more space economical than plain- \bar{I} storage, if $2 \leq d_{M,avg}$

Consequently, in case of cartesian materialized faceted taxonomies where the facets are hierarchically organized with $d_{M,avg} > 2$, TLOI requires $|Obj| * (d_{M,avg} - 2)$ less space. In cases

⁷Specifically, for MFTs where $d_{M,avg} > 2$

⁸Specifically, for a facet $F_i = (T_i, \leq_i)$ we traverse the graph defined by the following set of edges: $\leq_i \cup I_{|T_i}$ where $I_{|T_i}$ denotes the restriction (of the domain) of I on T_i .

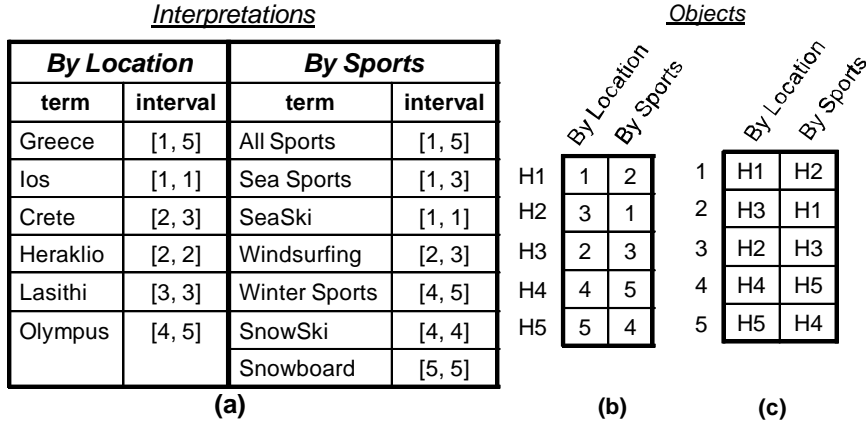


Figure 5.7: Store Indices

that $d_{M,avg} < 2$, the overhead of storing the $\bar{I}(t), \forall t \in \mathcal{T}$ will not be big so we can follow the plain- \bar{I} storage. Furthermore, an other major advantage of TLOI is that the storage overhead is independent to the depth of the hierarchies.

5.2.2.2 Indices for storing the taxonomies

According to section 3.2.3, in case we follow a maximal storage policy, we only need to store $\forall t \in \mathcal{T}$ the $B(t), N(t)$. So $\forall t \in \mathcal{T}$, we will store as sets the $B(t), N(t)$ and its interval which represents the $\bar{I}(t)$.

In case that $\forall t \in \mathcal{T}, I(t) \neq \emptyset$, these intervals allow us to check term subsumption in constant time, i.e. by checking if the interval of a term covers the interval of the other. For example if $t_{interval} = [2, 4]$ and $t'_{interval} = [2, 10]$ then it follows that $t < t'$. This means that if TLOI is adopted and $\forall t \in \mathcal{T}, I(t) \neq \emptyset$ then we do not have to label taxonomies (remember CTCA-based labeling optimization). We have to stress that TLOI is different from an "object-extended" application of the Agrawal's labeling, i.e. from the labeling obtained by considering each object o as "narrower term" of the terms in $D(o)$. This is evident in the example of Figure 5.8 which shows the derived labels by each approach. Notice that Agrawal's labels encode information of both terms and objects, while TLOI only of objects. For the problem at hand, TLOI is more appropriate because although it has the same complexity as Agrawal's labels (two integer comparisons), the label of a term t allows us to compute the $|\bar{I}(t)|$ in constant time as it is the size of its interval, e.g if $t_{interval} = [o_{start}, o_{end}]$ then $|\bar{I}(t)| = o_{end} - o_{start} + 1$.

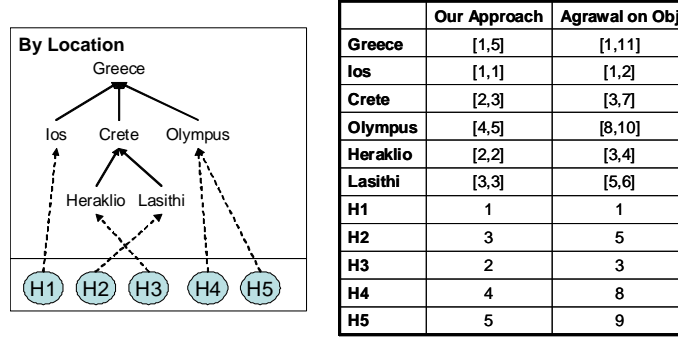


Figure 5.8: TLOI vs Object-extended Agrawal's Labeling

5.2.2.3 Zoom-in points Computation

Let us now see how we can compute $\bar{I}(ctx)$ in case a TLOI is available. To compute $\bigcap \{\bar{I}(ctx_i) \mid ctx_i \in ctx\}$ we can start from the term $t_{min} \in ctx$ with the minimum in size interval $invl_{min} = \min_{i=1}^k (|\bar{I}(ctx_i)|)$. For each $id \in invl_{min}$ we get the object o which the id belongs. Then, we get the set of identifiers (ids) of o for the rest facets. What we have to check is that each of these identifies belongs to the interval of the corresponding term of ctx . For example, let us assume that $ctx = \{Crete, SeaSports\}$. At first we get the intervals of these two terms, i.e. $[2, 3], [1, 3]$. The term *Crete* has the minimum in size interval, so $invl_{min} = [2, 3]$. Then, we get the object *H3* as it has the id 2 for the facet *By Location*. Next, we get its id for the facet *By Sports*, it is 3, and we check if it belongs to the interval $[1, 3]$. Yes, so $H3 \in \bar{I}(ctx)$. We do the same for id 2, and finally, the intersection is $\{H2, H3\}$.

The time complexity for computing $\bar{I}(ctx)$ will be the cost of getting the intervals of all terms $t \in ctx$ and the cost for computing their intersection using the method described earlier. The cost for getting the intervals is $O(k)$ as each term keeps its interval and the maximum number of the terms that ctx can have is k . The cost for computing the intersection will be $O(inv_{min} * (k - 1))$ as $\forall id \in invl_{min}$ we have to check the other $k - 1$ ids and the indices can be hash-based. So, the overall time complexity will be $O(k + invl_{min} * (k - 1))$. As the cost depends on the $invl_{min}$, this approach is not efficient in the following cases: (i) for foci whose terms belong to the highest levels of the hierarchy, as the $invl_{min}$ will be large, and (ii) the cardinalities of the interpretations of all terms $ctx_i \in ctx$ are the same. However, as FDT is an information thinning technique, and the *TLOI-based* approach is followed under specific situations e.g. under a threshold, these cases do not appear frequently.

Finally, to compute the zoom-in points, we can follow exactly the algorithms presented in

chapter 3. The advantage of TLOI is that minimizes the costs of computing the intersections.

5.2.2.4 TLOI Advantages

Until this point, we saw how TLOI can be used for computing efficiently the zoom-in points. However, by using a simple hash-based index for storing the interpretations, we can have the same efficiency. Both indices can check if an object $o \in \bar{I}(t)$ where $t \in \mathcal{T}$, in constant time. In case of a hash-based index, it is the cost of a look-up operation, while in case of TLOI, it is the cost on an interval enclosure check. Bellow, we summarize the advantages of TLOI that make it more efficient:

- TLOI is more space economical. As we prove in section 5.2.2.1, TLOI is $d_{M,avg} - 2$ times more space economical than plain- \bar{I} storage in cartesian materialized faceted taxonomies.
- TLOI can speedup up and other on-line tasks. We get the $|\bar{I}(t)|$ for a term $t \in \mathcal{T}$ in constant time. We need the count information for each term for two reasons: (i) computing approximately the focus size and (ii) finding which $ctx_i \in ctx$ has the minimum in size $|\bar{I}(ctx_i)|$.
- In case that $\forall t \in \mathcal{T}, I(t) \neq \emptyset$, TLOI allows us to check term subsumption in constant time.

Some other details regarding TLOI follow. If an object is not classified with respect to a facet, then this object will not get an id for that facet. If for a term $t \in \mathcal{T}_i$ the set $\bar{I}(t) = \emptyset$ then $t_{interval} = [-1, -1]$ and we do not take it into consideration.

Finally, we have to mention that TLOI approach can be used also for storing the \mathcal{P} parameters in *CTCA – based* approach.

5.2.2.5 TLOI on DAG and Multiple Classification

TLOI can also be used on DAG-structured taxonomies or on materialized faceted taxonomies with multiple classification of objects. In this section we examine the TLOI behavior on these particular cases.

For DAG-structured taxonomies we can follow the Agrawal’s approach on labeling graphs with the only difference that the optimal spanning tree ST will be selected with respect to the number of the objects that will be labeled. Then for each node of ST , the interval of

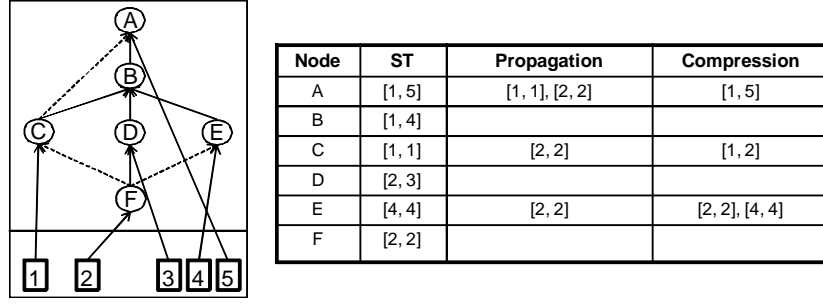


Figure 5.9: TLOI on DAG

source node is propagated to the target node and recursively up to its ancestors. After the step of propagation, a node may contain more than one interval so we need a step of intervals' compression. In case the intervals are adjacent, they can be merged. If an interval is subsumed by another, it can be pruned. Finally, after the execution of the two above steps, the node gets the remainder intervals. Figure 5.9 depicts an example of TLOI on a DAG. All the arrows which connect the DAG's nodes belong to DAG while only the non-dashed arrows belong to ST . The table shows the nodes' labels according to the spanning tree (column ST), the third column shows the labels at the propagation step and the last shows the compression step.

As Agrawal's labeling scheme on DAGs, the total storage requirement depends on the nature of the graph. In the worst case, the storage required for the compressed closure can be $O(|\mathcal{T}|^2)$, as in the case of a bipartite graph.

As we have already mentioned, in case of multiple classification, TLOI is not efficient. This is due to the facet that an object will have more than one ids for a facet. Figure 5.10(a) shows a tree-structured facet with three indexed objects $\{o_1, o_2, o_3\}$ where each object is classified under two terms of the facet. Figure 5.10(b) presents the objects' ids according to the DFS, and figure 5.10(c) shows the terms' labels.

5.2.3 Changes over Materialized Faceted Taxonomy

Taxonomy or objects' indexing updates may turn a CTCA expression e or TLOI indices invalid. In this section we examine how we can revise e and TLOI indices after a taxonomy or an object update.

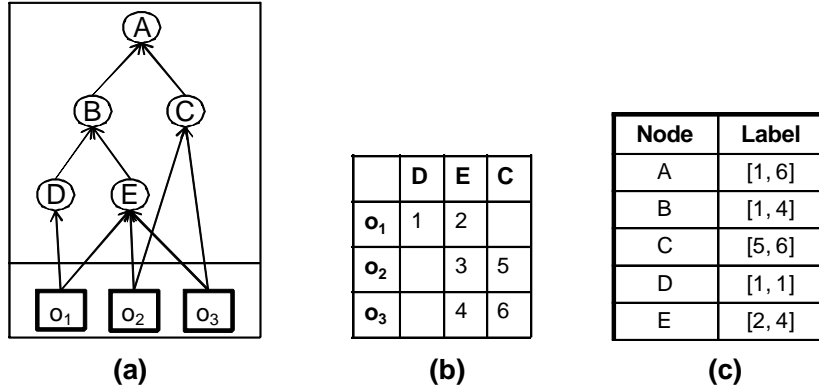


Figure 5.10: TLOI with multiple classification

5.2.3.1 CTCA Updates

CTCA allows the specification of the valid and invalid compound terms of a materialized faceted taxonomy according to the current state of affairs. Obviously, if the state of affairs changes, currently valid terms may become invalid and vice-versa. [52] shows how we can revise e after a taxonomy update⁹ and reach a valid expression e' whose semantics (compound terms defined) is as close as possible to the semantics of the original expression e before the update.

Furthermore, as a CTCA optimization uses the Agrawal's labeling scheme, we need to present how the labels which are assigned to each term will change. According to [4], to support incremental updates without node relabeling one can leave gaps between the intervals generated during the bottom-up tree traversal using some constant factor c in the postorder numbering, i.e., the label of a node u is $[index(u), c \times post(u)]$.

In case of objects' indexing updates, it is obvious that the expression e and its semantics may need to be changed. To support the $feed(A)$ operation also requires revising the expression e and this is a topic that is worth further research.

5.2.3.2 TLOI Updates

Lets examine first the case of taxonomy update i.e term deletion, addition. In case of a term t deletion, we need to determine if we also want to delete the objects which are directly classified under t i.e., $I(t)$ ¹⁰. In case we do not want to delete the $I(t)$ and we assume that they will be classified under t 's parents, the deletion is trivial as $\bar{I}(t') \supseteq \bar{I}(t)$ where $t' \in B(t)$. On the other

⁹The update operations are: term renaming, term deletion, term addition, subsumption relationship deletion, subsumption link addition, leaf addition and intermediate term addition.

¹⁰We assume that we want to delete only t and not $N^*(t)$

hand, in case we want to delete the $I(t)$, we must delete t and then follow the object deletion approach which will be presented below.

In case of term addition, we add t in a location specified by the user and then t 's label will be the union of its children intervals.

Furthermore, we have objects indexing updates i.e., $feed(A)$ operations. In case we want to delete an object $o \in Obj$, we simply delete o from Obj and we decrease the ids of all objects where their ids are greater than o id. Let Obj_{grt} denotes these objects i.e., $Obj_{grt}(o) = \{ob \in Obj \mid ob.id > o.id\}$. Subsequently, we relabel only the terms $t' \in \bar{D}(Obj_{grt}(o) \cup \{o\})$. It is obvious that the greater the id of o is, the less expensive the cost of deletion is going to be. As we use DFS algorithm, the best case of object deletion is the deletion of objects which belong to the extension of terms at the higher levels of hierarchy.

In case of object addition, we can follow two possible approaches. Let us assume that we want to add the object o where $D(o) = \{t\}$. According to the first approach, o gets the greatest id and we add to t an additional interval with the id of o . For example, if $|Obj| = 100$ the id of o will be 101 and the interval $[101, 101]$ will be added to the label of t . However, this approach is not so efficient as a term will have more than one intervals. The second approach avoids this shortcoming. Let us assume that the label of t is $[90, 95]$. Then, o will get the id 96 and the ids of the objects $ob \in Obj_{grt}(o)$ will be increased. Consequently, the label of t will be $[90, 96]$ and we have to relabel the terms $t' \in \bar{D}(Obj_{grt}(o) \cup \{o\})$. The cost of addition in this case is the same with the cost of deletion.

5.3 Evaluation

Section 5.3.1 compares analytically [5], [38] and *TLOI-based* approaches with respect to the storage overhead. Section 5.3.2 reports experimental results regarding the computation of \bar{I} using the *TLOI-based* approach. Finally, section 5.3.3 presents experimental results for the *CTCA-based* approach.

All algorithms are implemented in Java and experiments are performed on an ordinary PC (AMD Opteron 2,4GHz with 8GB RAM).

5.3.1 Analytical Evaluation

In this section we compare analytically from the storage point of view, the indices proposed in *TLOI-based* approach with the other object-based indices presented in section 3.2.2 ([38] and [5]).

Suppose that we have a cartesian materialized faceted taxonomy with similar characteristics as those presented in the global-scale exploration scenario (see section 5.1). In more detail, we have 5 facets, each one is hierarchically organized as a balanced and complete tree with degree 16 and depth 4. So, $|\mathcal{T}| \approx 3.5 * 10^5$. Furthermore, we suppose that $|Obj| = 10^{10}$ where we assume that the majority of objects are classified under the leafs of the trees, so $d_{M,avg} = 3.8$. This is a reasonable assumption as most objects are described by the most specific terms. In addition, we assume that the size of an integer (Int) is 4 bytes.

If we follow the approach proposed in [5], we have to store the *TaxIndex* and the inverted list that keeps the $\bar{I}(t)$ for each $t \in \mathcal{T}$ (see Figure 3.4). In this scenario we will not store the list which contains the objects's descriptions. *TaxIndex* keeps 2 integers for each $t \in \mathcal{T}$, so the required storage is $|\mathcal{T}| * 2 * \text{Int}$. For storing \bar{I} , we need $|Obj| * d_{M,avg} * C_M * \text{Int}$ bytes. In our case $C_M = k$ where k is the number of the facets.

If we follow the approach proposed in [38] we need 4 indices for storing the taxonomy (*SF*, *FS*, *Descendants* and *Ancestors*), while the interpretations will be stored in bitmaps and inverted lists. In more detail, for *SF* index we need

$$\left(\sum_{i=1}^d (b^i)\right) * \text{Int} * k$$

bytes, where d is the depth and b is the degree of the tree. For *FS* we need

$$\left(\sum_{i=0}^{d-1} (b^i)\right) * b * \text{Int} * k$$

bytes. For *Descendants* index we need $|\mathcal{T}| * AD * \text{Int}$ bytes, where AD is the average number of descendants of a term in the tree i.e. $AD = \frac{\sum_{i=0}^{d-1} (\sum_{k=i+1}^d (b^k))}{\sum_{i=0}^d (b^i)}$. On the other hand, storage overhead of *Ancestors* index is $|\mathcal{T}| * AA * \text{Int}$ where AA is the average number of ancestors of a term of the tree i.e. $AA = \frac{\sum_{i=1}^d (b^i * i)}{\sum_{i=0}^d (b^i)}$. For the reason that the facets' hierarchies are balanced and complete trees $AA = AD$.

Furthermore, the storage requirements of a bitmap-based interpretation according the author is $|\mathcal{T}| * |Obj|/8$, while an inverted list-based will require $|Obj| * d_{M,avg} * C_M * \text{Int}$. Suppose that

we will use a bitmap for storing the interpretations of the terms at the first level of the hierarchy and inverted lists for the rest. We will have $(16 * |Obj|/8) + |Obj| * (d_{M,avg} - 1) * C_M * \text{Int}$.

If we follow the *TLOI-based* approach, the storage overhead will be $2 * |Obj| * k * \text{Int}$ for storing the interpretations. For storing the hierarchies, we need $|\mathcal{T}| * 2 * \text{Int}$ for the intervals and the same storage overhead of [38] for storing the *SF, FS* indices, as $\forall t \in \mathcal{T}$ we store the $B(t), N(t)$.

Table 5.3 shows the sizes. As we can see, our approach is more space economical. In case that $d_{M,avg}$ will be increased, then the difference between storage overheads will be increased too. This is for the reason that our approach is independent to the structure of the facets' hierarchies.

Storage	[38]	[5]	our
\mathcal{T}	13 MB	2.6 MB	5.26 MB
Interpretations	0.54 TB	0.70 TB	0.37 TB

Table 5.3: Comparison table according Global Index Scenario

5.3.2 Computation of \bar{I} : Time Perspective

In this section we compare two different \bar{I} evaluation approaches with respect to the time required for computing the $\bar{I}(ctx)$. In more detail, we compare the *TLOI* algorithm with the *Extension Interpretation-based* evaluation approach, following the *maximal storage* policy. In the experiments the $\bar{I}(t), \forall t \in \mathcal{T}$ was stored in inverted lists and they were not stored in hash-based indices. We did not provide experiments on bit-map indices but we discuss times that have been presented in [39] and we compare them with the inverted lists approach.

We did not make experiments in a billion object information base because, apart from not having such an information base and according to our approach the computation of zoom points is based on the extensions only if the focus size is under a threshold. In particular, we created a faceted taxonomy of 10 hierarchically organized facets and an information base of 10^6 objects. The $|\bar{I}(ht)|$ of a term ht at the highest level of hierarchy (root element) was 10^6 objects, for a term mt at the middle levels was $|\bar{I}(mt)| = 1,25 * 10^5$ objects, while for a low level term lt was $|\bar{I}(lt)| = 10^4$.

We computed the \bar{I} for 6 different types of ctx :

- i) all terms were ht (HT),

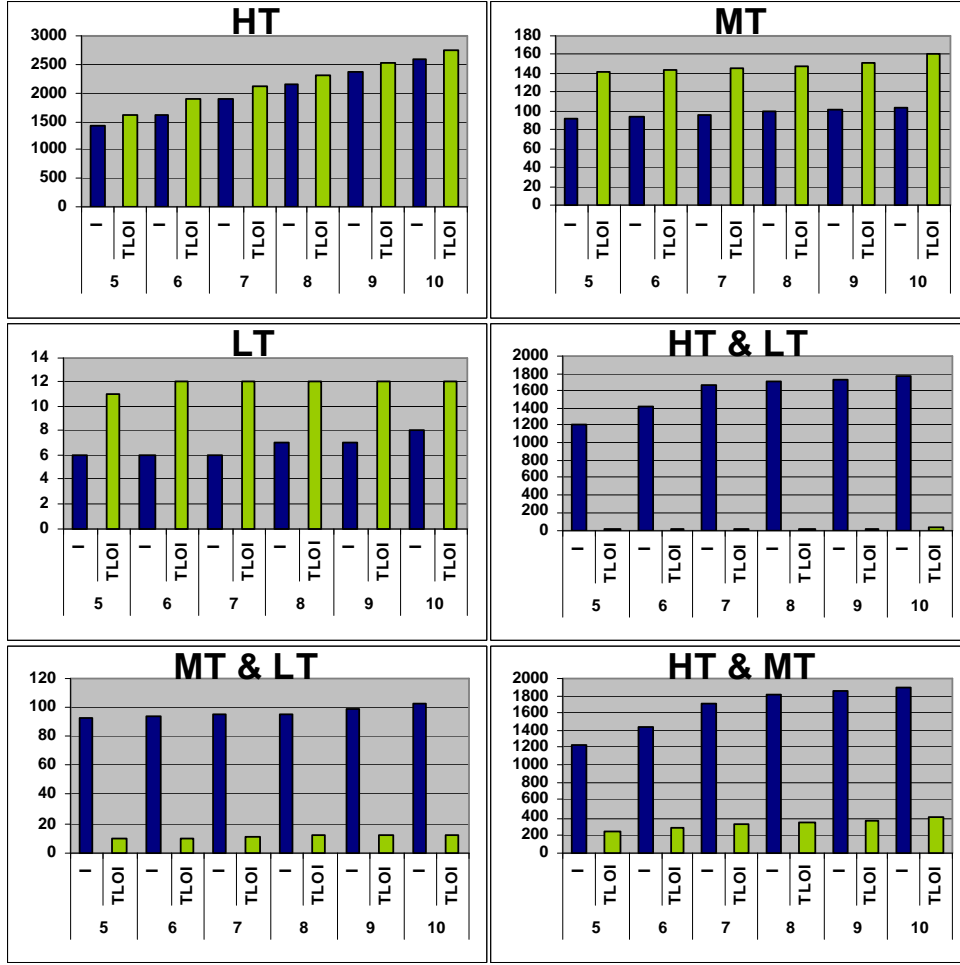


Figure 5.11: $\bar{I}(ctx)$ Measures

- ii) all terms were *mt* (MT),
- iii) all terms were *lt* (LT),
- iv) one term was *lt* and the rest were *ht* (HT & LT),
- v) one term was *lt* and the rest were *mt* (MT & LT), and
- vi) one term was *mt* and the rest *ht* (HT & MT).

Each of the above foci, was executed for 5,6,...,10 facets. Figure 5.11 shows the execution times in milliseconds where each execution time is the average of 10^3 different executions.

As we can see from Figure 5.11, the times are proportional to the number of facets because as $|ctx|$ increases, the number of intersections increases as well.

Regarding the comparison TLOI vs classical *Extension Interpretation-based* evaluation approach, we can see that if all ctx_i have the same $|\bar{I}|$ or belong to the same hierarchical level, *Extension Interpretation-based* is faster than TLOI approach, e.g in cases (HT), (MT) and (LT).

For example, in (MT) case with 5 facets, \bar{I} method takes 92ms, while TLOI takes 141ms. However, this result was expected as we have already discussed in section 5.2.2. On the other hand, in case we have a *ctx* with terms in different hierarchical levels e.g. (HT & LT), (MT & LT) and (HT & MT), TLOI is much faster. For example, in case we have 10 facets and all terms are in the highest level except one which is in the lowest, using \bar{I} approach we need 1770 ms while with TLOI we need only 32 ms. This means that TLOI is around two orders of magnitude faster. This is an expected result as TLOI checks the *ids* of 10^4 objects while interpretation-based approach makes intersections between sets of 10^6 integers.

Sacco in [39], presented several experiments for determining which implementation between inverted lists and bitmap is more efficient. His experiments showed that the bitmap-based approach clearly outperforms the inverted lists-based. In more detail, he showed that in a corpus of $8 \cdot 10^5$ objects which are classified under 10 hierarchically organized facets, the bitmap-based implementation is about 35% faster than the inverted lists-based, while when the size of collection decreased also decreased the difference between the two implementations. In case that the size of the collection is less than 10^5 , the inverted file-based implementation is more efficient. However, this approach can be followed only for a focus *ctx* where all terms $t \in ctx$ also belong to the high levels of the taxonomy¹¹. In this case, as is barely the focus size be under the *thres*, we follow the *CTCA-based* approach.

5.3.3 CTCA Validity Checking

In this section we present experimental results on *CTCA-based* approach. In more detail, section 5.3.3.1 estimates the number of \mathcal{P}, N parameters in a terra-sized collection. Section 5.3.3.2 presents execution times of *isValid* algorithm without the usage of any optimization. Section 5.3.3.3 presents experimental results on *isValid* algorithm by using the labeling taxonomies optimization presented in section 5.2.1.6 and compares it with the *FDT-based CTCA's parameters indexing* optimization presented in the same section.

5.3.3.1 Estimation of CTCA Parameters Plurality

A general discussion about the size of the parameters is given in [51]. In our case, we have to take into consideration the organization of the facets, e.g. whether they are hierarchically

¹¹According to [38], in bitmap are stored only the interpretations of the high level terms.

organized or flat, and the size of the collection i.e. $|Obj|$. We will elaborate on cases where the facets are hierarchically organized, as in our examples and experiments there is no any flat facet, and we have a terra-sized collection of data.

There are three main cases according to the size of hierarchies (wrt the $|\mathcal{T}|$):

Case 1 (Small Taxonomies). As the number of all possible compound terms is not big and the collection is very large, we need a minus-product operation with few $n \in N$ parameters to describe the $V^c(M)$, where $V^c(M) = \{T_1 \times \dots \times T_k\} \setminus V(M)$.

Case 2 (Very Large Taxonomies). Here the number of all possible compound terms is much bigger than the distinct descriptions of the collection's objects. Therefore a plus product having a parameter P where $P = \min_{\preceq}(V(M))$, is beneficial.

Case 3 (Medium-sized Taxonomies). This is a case that falls between the above two extremes. In such cases it is beneficial to use a plus-product operation if

$$|\min_{\preceq}(V(M))| < |\max_{\preceq}(V^c(M))|, \text{ or a minus-product otherwise.}$$

If e has several plus and minus product operations, the range-restricted closed world assumptions of these operations make hard the estimation of the $|\mathcal{P} \cup \mathcal{N}|$. The exact estimation of CTCA parameters' plurality is an issue which worths further research.

5.3.3.2 isValid Experiments Without Optimizations

In this section, we present experiments for cases that fall between *Case 2*, *Case 3*. We do not use any optimization like these presented in sections 5.2.1.6. We created three types of expressions with the following formats:

e1: $\oplus_P(e_1, \dots, e_k)$,

e2: $\ominus_N(e_1, \dots, e_k)$, and

e3: $\ominus_N(\oplus_P(\ominus_N(\oplus_P(e_1, e_2), e_3, \dots, e_k))$.

For each expression we created different \mathcal{P} and \mathcal{N} sets with the following cardinalities: 10^3 , 10^4 and 10^5 . For the expressions of the e3 format, the number of parameters for each one subexpression was $|\mathcal{P} \cup \mathcal{N}|/(k - 1)$.

As we did not apply any labeling scheme, we had to consider the characteristics of facets' organization. We created 4 different faceted taxonomies each one consisted of 5 facets. In all faceted taxonomies the facets were organized as a balanced and complete tree with *depth* = 5. The only difference between faceted taxonomies was the degree of the facets' trees. In the first one it was 3, for the second it was 4, 6 for the third, and 10 for the last. Additionally, $\forall t \in \mathcal{T}$

we stored the $N(t)$ and $B(t)$.

The compound terms of each parameter had average term depth=3 and each P , N was redundancy free as $P = \min_{\preceq}(P)$ and $N = \max_{\preceq}(N)$. Table 5.4 presents the execution times, where each one is in milliseconds and is the average of 10^4 different executions.

Degree	Num. of Params	$e1$	$e2$	$e3$
3	10^3	14	15	8
	10^4	144	156	87
	10^5	1413	1566	861
4	10^3	40	41	22
	10^4	399	406	239
	10^5	3979	4100	2367
6	10^3	301	328	206
	10^4	3132	3406	1975
	10^5	32209	33804	19741
10	10^3	12357	12345	8848
	10^4	128446	133989	82621
	10^5	1292457	1350079	803020

Table 5.4: *isValid* execution times without optimizations

The results, proves that the usage of *isValid* algorithm without any optimization is prohibitive.

5.3.3.3 isValid Experiments With Optimizations

In this section we do not present experimental results on the *FDT-based CTCA's parameters indexing* optimization as the execution times presented in section 5.3.2 covers this case. In case of plus product operations, we can see that execution times using the TLOI approach, while in case of minus product operation the times of \bar{I} . We cannot use the TLOI method in minus product operations, as for storing the \underline{Y} , each term $t \in \mathcal{T}$ will have several labels.

Below, we present experiments on the *labeled taxonomies and naive method of indexing* optimization approach, and finally we compare the two optimizations.

Firstly, we defined a faceted taxonomy of 10 facets each having the form of a balanced and complete tree with depth = 5 and degree = 6. The taxonomies were labeled using the Agrawal's labeling scheme. We believe that 10^6 parameters are enough for a terra-sized collection with the above characteristics. We executed the *isValid* algorithm for foci with $|ctx| = 5, 6, \dots, 10$. We used the same format expressions as in the experiment presented above (section 5.3.3.2).

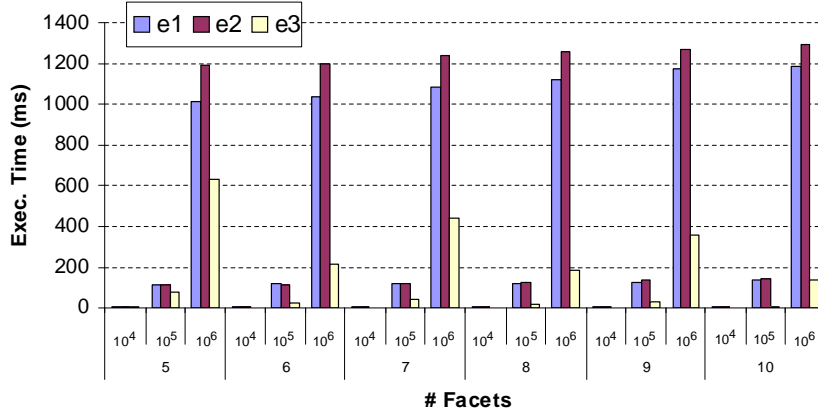


Figure 5.12: *isValid* execution times with terms labeling optimization

Figure 5.12 shows the execution times where axis x contains the cardinalities of parameters e.g. 10^4 , 10^5 , 10^6 and the number of facets.

We can see that in general, expressions of the $e2$ format take more time to be evaluated as *isValid* has one more operation to execute (line 18 of *isValid* algorithm) than $e1$. Additionally, for expressions of $e1, e2$ format, the execution time of *isValid* algorithm was proportional to the number of the facets. This is reasonable as we have to check one more interval for each parameter.

As regards the expressions of $e3$ format, it is easy to see that the evaluation is faster. This happens because we have subexpressions, so we firstly search the parameters of an expression and if there is no a suitable parameter then we look for a suitable to the parameters of the subexpressions. In cases of 6, 8 and 10 facets the time is less than having 5, 7, or 9 as the expressions start with a plus-product operation while in the rests start with a minus. Furthermore, the execution time of *isValid* is decreased while the number of the facets is increased. This is a reasonable result as the size of P or N of a subexpression is degreased as the size of parameters for each subexpression was $|\mathcal{P} \cup \mathcal{N}|/(k-1)$.

Optimizations Comparison

Firstly, the naive method is more space economical than using FDT-based as it does not need any special index. But let us assume that we have not any space limitation.

According to the Figure 5.11 and Figure 5.12, a hybrid approach for checking if ctx is valid seems to be better. This hybrid approach has to take into consideration the format of the expression. In case of $e1$ and $e2$ formats, it has to take into account the size of each $\bar{Y}(ctx_i)$ and

$\underline{Y}(ctx_i)$ respectively. Then, according to the time that FDT-based approach needs to compute such interpretations it will decide the method to follow. For example, in case we have 10 facets, all terms $ctx_i \in ctx$ have the same $|\bar{I}| = 10^6$, and e has the $e1$ format, the FDT-based needs 2574ms while naive method needs only 1300ms. On the other hand, in case we have the same characteristics but all terms $ctx_i \in ctx$ have $|\bar{I}| = 10^6$ except one which has 10^4 , FDT-based approach will need only 32ms while naive method will also need 1300ms. It is clear that in first case we will use the naive method while in second case the FDT-based. Here we have to mention that in case of $e1$ it will use the TLOI method, while in case of $e2$ only the *Extension Interpretation-based* evaluation approach. In case of $e3$, we will follow the above approaches recursively.

It is obvious that using the above hybrid approach and according to the experimental evaluation, the proposed approaches are efficient enough for such magnitudes.

Chapter 6

Conclusion and Future Work

This thesis introduced a framework for facet-based exploration services and described in detail the engineering aspects of supporting such services. The contribution of this thesis lies in:

- studding analytically a number of possible architectures that one can follow to develop a faceted exploration application with respect to the available resources i.e RAM, and the size of the collection. This study included the description of several algorithms (for enabling such services) and their complexity, the description of several architectures, the identification of particular query plans (for achieving efficiency), the description of their applicability and finally the presentation of comparative experimental results over large data sets.
- implementing a Main Memory API called **flexplorer**, written in Java which provides the core functionality for implementing the faceted exploration model. **flexplorer** can be exploited in various ways. For instance, a human user (developer) could use this framework to define the desired facets and taxonomies or for importing existing taxonomies. Additionally, a user could use it for providing faceted access to a corpus of metadata records or to a structured source. Moreover, it could be used by tools that mine facets and terms, e.g. [11, 13, 25], or tools that create automatically the faceted metadata structures, e.g [47]. In general, we could say that such a framework can serve as the middleware between the presentation layer and the underlying information sources.
- developing various applications e.g. in Mitos web search engine and European Space Agency web portal which use the API,

- developing and presenting the dynamic coupling of results clustering with dynamic faceted taxonomies resulting to an effective, flexible and efficient exploration experience and
- making an investigation on various techniques that could be used for advancing the scalability of such services.

Directions that are worth further research include:

- **Regarding Terra-sized collections**

The proposed indices for storing CTCA's parameter do not support $feed(A)$ operations. So, the user has not the ability of exploration on a specific set of objects. We plan to elaborate on CTCA-based approach, and propose indices and algorithms for supporting this type of operation.

Furthermore, we plan to elaborate on CTCA parameters plurality estimation. We need a formula which will take as input a materialized faceted taxonomy and will export a minimum and maximum number of CTCA parameters.

- **Regarding FDT general architectures**

Elaborating on DB-MEM architecture. This will be a hybrid approach. Data will be stored in a DB, however some of them will be kept in main memory. For instance, the hierarchically organized attributes values could be kept in Main Memory while the rest in a DBMS. The benefits of this approach is that the hierarchies are loaded once, and the SQL queries that are sent to the DBMS are more simple and faster to execute.

Moreover, we plan to compare experimentally all algorithms presented in section 3.2.3.1 for both (MEM) and (DB) architectures.

- **Regarding facets for web searching**

Provide to users the ability to define their own taxonomy. In this case any pair (n, q) where n is a user-provided name and q is any **Mitos**-query could be considered as a term.

Bibliography

- [1] Online Computer Library Center. Dewey decimal classification. Available at www.oclc.org/dewey.
- [2] “XFML: eXchangeable Faceted Metadata Language”. <http://www.xfml.org>.
- [3] Special issue on Supporting Exploratory Search. *Communications of the ACM*, 49(4), April 2006.
- [4] R. Agrawal, A. Borgida, and HV Jagadish. Efficient management of transitive relationships in large data and knowledge bases. *ACM SIGMOD Record*, 18(2):253–262, 1989.
- [5] O. Ben-Yitzhak, N. Golbandi, N. Har’El, R. Lempel, A. Neumann, S. Ofek-Koifman, D. Sheinwald, E. Shekita, B. Sznajder, and S. Yogev. Beyond basic faceted search. *In WSDM ’08*, pages 33–44, 2008.
- [6] K. Chakrabarti, S. Chaudhuri, and S. Hwang. “Automatic Categorization of Query Results”. *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data*, pages 755–766, 2004.
- [7] S. Chaudhuri and U. Dayal. An overview of data warehousing and OLAP technology. *ACM Sigmod Record*, 26(1):65–74, 1997.
- [8] V. Christophides, G. Karvounarakis, D. Plexousakis, M. Scholl, and S. Tourtounis. Optimizing taxonomic semantic web queries using labeling schemes. *Web Semantics: Science, Services and Agents on the World Wide Web*, 1(2):207–228, 2004.
- [9] D.H. Cunningham, D.D. Maynard, D.K. Bontcheva, and M.V. Tablan. GATE: A framework and graphical development environment for robust NLP tools and applications. 2002.

- [10] D.R. Cutting, D. Karger, J.O. Pedersen, and J.W. Tukey. Scatter/Gather: A cluster-based approach to browsing large document collections. In *Procs of the 15th Annual Intern. ACM Conf. on Research and Development in Information Retrieval, (SIGIR'92)*, pages 318–329, Copenhagen, Denmark, June 1992.
- [11] W. Dakka, R. Dayal, and P.G. Ipeirotis. “Automatic Discovery of Useful Facet Terms”. *SIGIR Faceted Search Workshop*, Aug. 2006.
- [12] W. Dakka and P.G. Ipeirotis. Automatic extraction of useful facet hierarchies from text databases. In *Procs of the 24th Intern. Conf. on Data Engineering, (ICDE'08)*, pages 466–475, Cancún, México, April 2008.
- [13] Wisam Dakka, Panagiotis G. Ipeirotis, and Kenneth R. Wood. “Automatic Construction of Multifaceted Browsing Interfaces”. In *Procs of the 14th ACM CIKM '05*, pages 768–775, New York, NY, USA, Nov. 2005.
- [14] Sebastien Ferre and Olivier Ridoux. Logical information systems: from taxonomies to logics. In *Procs of FIND'2007 (at DEXA '07)*, pages 212–216, Regensburg, Germany, 3-7 Sept. 2007.
- [15] H.B. Frej, P. Rigaux, and N. Spyrtos. User notification in taxonomy based digital libraries. In *Intl. Symposium on Design of Communication*, 2006.
- [16] Bernhard Ganter and Rudolf Wille. “*Formal Concept Analysis: Mathematical Foundations*”. Springer-Verlag, Heidelberg, 1999.
- [17] F. Giunchiglia, B. Dutta, and V. Maltese. “Faceted Lightweight Ontologies”. Technical Report DISI-09-022, University Of Trento - Dipartimento Di Ingegneria E Scienza Dell' Informazione, April 2009.
- [18] R.G. Gordon, B.F. Grimes, and Summer Institute of Linguistics. *Ethnologue: Languages of the world*. SIL International Dallas, TX, 2005.
- [19] A. Gulli and A. Signorini. The indexable web is more than 11.5 billion pages. In *International World Wide Web Conference*, pages 902–903. ACM New York, NY, USA, 2005.

- [20] M.A. Hearst and J.O. Pedersen. Reexamining the cluster hypothesis: Scatter/Gather on retrieval results. In *Procs of the 19th Annual Intern. ACM Conf. on Research and Development in Information Retrieval, (SIGIR'96)*, pages 76–84, Zurich, Switzerland, August 1996.
- [21] Michiel Hildebrand, Jacco van Ossenbruggen, and Lynda Hardman. “/facet: A Browser for Heterogeneous Semantic Web Repositories”. In *Procs of ISWC '06*, pages 272–285, Athens, GA, USA, Nov. 2006.
- [22] Eero Hyvönen, Eetu Mäkelä, Mirva Salminen, Arttu Valo, Kim Viljanen, Samppa Saarela, Miikka Junnila, and Suvi Kettula. “MUSEUMFINLAND - Finnish Museums on the Semantic Web”. *Journal of Web Semantics*, 3(2-3):224–241, 2005.
- [23] Eero Hyvonen, Eetu Mdkeld, Mirva Salminen, Arttu Valo, Kim Viljanen, Samppa Saarela, Miikka Junnila, and Suvi Kettula. “MuseumFinland – Finnish Museums on the Semantic Web”. *Journal of Web Semantics*, 3(2):25, 2005.
- [24] Amy K. Karlson, George G. Robertson, Daniel C. Robbins, Mary P. Czerwinski, and Greg R. Smith. “FaThumb: a Facet-Based Interface for Mobile Search.”. In *Procs of the Conference on Human Factors in computing systems, CHI'06*, pages 711–720, New York, NY, USA, Apr. 2006.
- [25] C. Kohlschütter, P.A. Chirita, and W. Nejdl. Using Link Analysis to Identify Aspects in Faceted Web Search. In *SIGIR'2006 Faceted Search Workshop*, 2006.
- [26] B. Kules, M. Wilson, M. Schraefel, and B. Shneiderman. From keyword search to exploration: How result visualization aids discovery on the web. *Human-Computer Interaction Lab Technical Report HCIL-2008-06, University of Maryland*, pages 2008–06, 2008.
- [27] E. Makela, K. Viljanen, P. Lindgren, M. Laukkanen, and E. Hyvonen. Semantic yellow page service discovery: The veturi portal. In *poster paper at ISWC '05*, Nov. 2005.
- [28] M. T. Maria Teresa Pazienza, editor. *Information Extraction: Towards Scalable, Adaptable Systems*, volume 1714 of *Lecture Notes in Computer Science*. Springer, 1999.
- [29] Eetu Mdkeld, Eero Hyvfnen, and Samppa Saarela. “Ontogator - A Semantic View-Based Search Engine Service for Web Applications.”. In *Procs of ISWC '06*, pages 847–860, Athens, GA, USA, Nov. 2006.

- [30] U. Y. Nahm and R. J. Mooney. Text mining with information extraction. In *Procs of AAAI 2002 Spring Symposium on Mining Answers from Texts and Knowledge Bases*, pages 60–67, 2002.
- [31] L. Nourine and O. Raynaud. A fast incremental algorithm for building lattices. *JETAI: Journal of Experimental & Theoretical Artificial Intelligence*, 14:217–227, 2002.
- [32] E. Oren, R. Delbru, and S. Decker. “Extending Faceted Navigation for RDF Data”. In *Procs of ISWC ’06*, pages 559–572, Athens, GA, USA, Nov. 2006.
- [33] P. Papadakos, S. Kopidaki, N. Armenatzoglou, and Y. Tzitzikas. Exploratory web searching with dynamic taxonomies and results clustering. In *ECDL ’09: Proceedings of the 13th European Conference on Digital Libraries*, Corfu, Greece, September 2009. (to appear).
- [34] P. Papadakos, Y. Theoharis, Y. Marketakis, N. Armenatzoglou, and Y. Tzitzikas. ”Mitos: Design and Evaluation of a DBMS-based Web Search Engine”. In *Procs of the 12th Pan-Hellenic Conference on Informatics (PCI’08)*, Greece, August 2008 (to appear).
- [35] P. Papadakos, G. Vasiliadis, Y. Theoharis, N. Armenatzoglou, S. Kopidaki, Y. Marketakis, M. Daskalakis, K. Karamaroudis, G. Linardakis, G. Makrydakis, V. Papathanasiou, L. Sardis, P. Tsialiamanis, G. Troullinou, K. Vandikas, D. Velegrakis, and Y. Tzitzikas. “The Anatomy of Mitos Web Search Engine”, <http://arxiv.org/abs/0803.2220>, Mar. 2008.
- [36] P. Pellegrin and A. Preus. *Aristotle’s classification of animals: biology and the conceptual unity of the Aristotelian corpus*. University of California Press, 1986.
- [37] S. R. Ranganathan. “The Colon Classification”. In Susan Artandi, editor, *Vol IV of the Rutgers Series on Systems for the Intellectual Organization of Information*. New Brunswick, NJ: Graduate School of Library Science, Rutgers University, 1965.
- [38] G. M. Sacco. Efficient implementation of dynamic taxonomies. Technical report, Univ. di Torino, Dip. di Informatica, 2004.
- [39] G. M. Sacco and Y. Tzitzikas. *Dynamic Taxonomies and Faceted Search: Theory, Practise and Experience*. Springer, 2009.
- [40] G. M. Sacco, Y. Tzitzikas, S. Ferre, P. G. Ipeirotis, W. Dakka, M. Stefaner, S. Perugini, Y. Zhang, and J. Koren. *Dynamic Taxonomies and Faceted Search: Theory, Practice and Experience*. Springer, 2009. ISBN: 978-3-642-02358-3.

- [41] Giovanni M. Sacco. “Dynamic Taxonomies: A Model for Large Information Bases”. *IEEE Transactions on Knowledge and Data Engineering*, 12(3), May 2000.
- [42] Giovanni Maria Sacco. “Guided Interactive Information Access for E-Citizens”. In *Procs. of the 4th Intern. Conf. on Electronic Government (EGOV-2005)*, pages 261–268, 2005.
- [43] G.M. Sacco. Some Research Results in Dynamic Taxonomy and Faceted Search Systems. In *SIGIR’2006 Workshop on Faceted Search*, 2006.
- [44] M.C. Schraefel, Maria Karam, and Shengdong Zhao. “mSpace: Interaction Design for User-Determined, Adaptable Domain Exploration in Hypermedia”. In *Procs of Workshop on Adaptive Hypermedia and Adaptive Web Based Systems*, pages 217–235, Nottingham, UK, Aug. 2003.
- [45] Oliver Sinnen. *Task Scheduling for Parallel Systems*. Wiley-Interscience, 2007.
- [46] E. Sperner. Ein satz über untermengen einer endlichen menge. *Mathematische Zeitschrift*, 27(1):544–548, 1928.
- [47] E. Stoica, M.A. Hearst, and M. Richardson. Automating Creation of Hierarchical Faceted Metadata Structures. In *Proceedings of NAACL HLT*, pages 244–251, 2007.
- [48] J. Teevan, S. Dumais, and Z. Gutt. Challenges for supporting faceted search in large, heterogeneous corpora like the Web. In *HCIR*, 2008.
- [49] A.K. Tsakalidis. Maintaining order in a generalized linked list. *Acta informatica*, 21(1):101–112, 1984.
- [50] D. Tunkelang. Faceted Search. *Synthesis Lectures on Information Concepts, Retrieval, and Services*, 1(1):1–80, 2009.
- [51] Y. Tzitzikas. An Algebraic Method for Compressing Symbolic Data Tables. *Journal of Intelligent Data Analysis*, 10(4), September 2006.
- [52] Y. Tzitzikas. Evolution of faceted taxonomies and CTCA expressions. *Journal of Knowledge and Information Systems*, 13(3):337–365, 2007.
- [53] Y. Tzitzikas and A. Analyti. Mining the Meaningful Term Conjunctions from Materialised Faceted Taxonomies: Algorithms and Complexity. *Journal of Knowledge and Information Systems*, 9(4):430–467, May 2006.

- [54] Y. Tzitzikas, A. Analyti, N. Spyratos, and P. Constantopoulos. An Algebraic Approach for Specifying Compound Terms in Faceted Taxonomies. In *EJC'03*, pages 67–87. IOS Press, 2004.
- [55] Y. Tzitzikas, A. Analyti, N. Spyratos, and P. Constantopoulos. An algebra for specifying valid compound terms in faceted taxonomies. *Journal of Data & Knowledge Engineering*, 62(1):1–40, 2007.
- [56] Y. Tzitzikas, N. Armenatzoglou, and P. Papadakos. FleXplorer: A Framework for Providing Faceted and Dynamic Taxonomy-based Information Exploration. In *Procs of FIND'2008 (at DEXA '08)*, Torino, Italy, Sept. 3, 2008.
- [57] Y. Tzitzikas, R. Launonen, M. Hakkarainen, P. Kohonen, T. Leppanen, E. Simpanen, H. Tornroos, P. Uusitalo, and P. Vanska. FASTAXON: A system for FAST (and Faceted) TAXONomy design. In *ER'04*, 2004.
- [58] Yannis Tzitzikas. “Revising Faceted Taxonomies and CTCA Expressions”. In *Proceedings of the 4th Hellenic Conference on AI, SETN 2006*, Heraklion, Greece, May 2006.
- [59] K.P. Yee, K. Swearingen, K. Li, and M. Hearst. Faceted Metadata for Image Search and Browsing. In *SIGCHI '03*, pages 401–408, 2003.
- [60] O. Zamir and O. Etzioni. Web document clustering: A feasibility demonstration. In *Procs of the 21th Annual Intern. ACM Conf. on Research and Development in Information Retrieval, (SIGIR'98)*, pages 46–54, Melbourne, Australia, August 1998.

Chapter 7

Appendix

7.1 Proofs

Prop. 1 (Lower Bound)

In a cartesian materialized faceted taxonomy, if $t_1, \dots, t_k \in \times_{i=1}^k T_i$, then

$$LB(|\bar{I}(t_1, \dots, t_k)|) = \max(0, \sum_{i=1}^k |\bar{I}(t_i)| - (k-1)|Obj|)$$

Proof:

We will prove this inductively. For $m = 1$ we have $LB(\bar{I}(t_1)) = \max(0, |\bar{I}(t_1)|) = |\bar{I}(t_1)|$ which is obviously true. Let assume that Prop. 1 holds for $k = m$ (where $m > 1$), i.e. suppose that it holds:

$$LB(|\bar{I}(\{t_1, \dots, t_m\})|) = \max(0, \sum_{i=1}^m |\bar{I}(t_i)| - (m-1)|Obj|).$$

What we have to prove is that it holds also for $k = m + 1$, i.e. to prove that:

$$LB(|\bar{I}(\{t_1, \dots, t_{m+1}\})|) = \max(0, \sum_{i=1}^{m+1} |\bar{I}(t_i)| - m|Obj|).$$

Let $ctx = ctx' \cup \{t_{m+1}\}$ where $ctx' = \{t_1, \dots, t_m\}$. Assuming single and mandatory classification, we can easily see (e.g. through a Venn diagram), that $|\bar{I}(ctx)| = |\bar{I}(ctx') \cap \bar{I}(t_{m+1})| \geq 1$ if and only if:

$$\begin{aligned} |\bar{I}(ctx')| + |\bar{I}(t_{m+1})| &\geq |Obj| + 1 \Leftrightarrow \\ |\bar{I}(ctx')| &\geq |Obj| + 1 - |\bar{I}(t_{m+1})| \end{aligned}$$

We can restate the above, and if X is a non negative integer we can write:

$$LB(|\bar{I}(ctx)|) = X \Leftrightarrow |\bar{I}(ctx')| \geq |Obj| + X - |\bar{I}(t_{m+1})|$$

If in the above inequality we replace $|\bar{I}(ctx')|$ by its lower bound (according to the inductive hypothesis) we get:

$$\begin{aligned}
LB(|\bar{I}(ctx)|) &= X \Leftrightarrow \\
|\bar{I}(ctx')| &\geq |Obj| + X - |\bar{I}(t_{m+1})| \Leftrightarrow \\
\sum_{i=1}^m |\bar{I}(t_i)| - (m-1)|Obj| &\geq |Obj| + X - |\bar{I}(t_{m+1})| \Leftrightarrow \\
\sum_{i=1}^{m+1} |\bar{I}(t_i)| - m|Obj| &\geq X
\end{aligned}$$

It follows that:

$$\begin{aligned}
LB(|\bar{I}(ctx)| = X &\Leftrightarrow \max(0, \sum_{i=1}^{m+1} |\bar{I}(t_i)| - m|Obj|) \geq X \\
LB(|\bar{I}(ctx)| &= \max(0, \sum_{i=1}^{m+1} |\bar{I}(t_i)| - m|Obj|)
\end{aligned}$$

□

scjsidcyneyxenrnwrwfdsw