

Information Visualisation and the Electronic Health Record

Visualising Collections of Patient Histories from General Practice

Stein Jakob Nordbø

Master of Science in Computer Science

Submission date: July 2006

Supervisor: Øystein Nytrø, IDI

Co-supervisor: Ole Edsberg, IDI
Anders Grimsmo, NSEP
Tom Christensen, NSEP

Problem Description

In the Norwegian health-care system, the general practitioner plays the role of a gate-keeper to more specialised health services. Also, the list patient system makes patient-practitioner relationships relatively stable. Because of these two factors, Norwegian general practitioner's databases often contain patient histories that are long and relatively complete. There are several, somewhat complementary, ways that knowledge can be extracted from these databases. 1) By reading the journal directly. 2) By statistical analysis. 3) By Knowledge-Discovery and Data mining (KDD). In this project, we are interested in a fourth way: By visualising information from the database in a way that enables the GP (or other user) to use his or her own visual processing system to get an overview and discover features in a collection of patient histories.

In this project, the candidate should design, implement and evaluate a prototype visualisation system for at least one type of patient histories, for example by doing the following (not necessarily in this order):

- * Select one or more medical problems, for example hypertension or hypothyreosis

For each problem:

- * Choose a visualisation technique that will be suitable for the problem.

- * Extract, filter and represent data for the problem.

- * Implement a visualisation prototype for the problem.

- * Empirically evaluate the usefulness of the chosen visualisation technique for this particular problem.

The candidate should consult both relevant literature and clinical experts while undertaking the project.

Assignment given: 20. January 2006

Supervisor: Øystein Nytrø, IDI

Preface

This is a Master's thesis written during the spring semester of 2006 at the Institute of Computer and Information Sciences (IDI) at the Norwegian University of Science and Technology (NTNU), for the Norwegian Centre of Electronic Health Records Research (NSEP). The supervisor has been Øystein Nytrø, while Ole Edsberg has been the main advisor. Other advisors have been Anders Grimsmo and Tom Christensen.

Acknowledgements

The author wishes to thank Ole Edsberg for his guidance and advice, and for all the interesting discussions we have had during this semester. Also, thanks to general practitioner and advisor Anders Grimsmo, who kindly participated in our case study, and supervisor Øystein Nytrø for encouraging feedback.

Summary

This thesis investigates the question:

How can we use information visualisation to support retrospective, explorative analysis of collections of patient histories?

Building on experience from previous projects, we put forth our answer to the question by making the following contributions:

- Reviewing relevant literature.
- Proposing a novel design for visual exploration of collections of histories, motivated in a specific problem within general practice health care and existing work in the field of information visualisation. This includes both presentation and interactive navigation of the data.
- Describing a query language and associated algorithms for specifying temporal patterns in a patient history.
- Developing an interactive prototype to demonstrate our design, and performing a preliminary case study. This case study is not rigorous enough to conclude about the feasibility of the design, but it forms a foundation for improvements of the prototype and further evaluation at a later stage.

We envision that our design can be instrumental in exploring experiences in terms of treatment processes. In addition, we believe that the visualisation can be useful to researchers looking at data to be statistically evaluated, in order to discover new hypotheses or get ideas for the best analysis strategies.

Our main conclusion is that the proposed design seems promising, and we will further develop our results through a research project during the summer and autumn of 2006.

Contents

1	Introduction	1
1.1	Motivation	1
1.2	Our approach to the problem	3
1.3	Case to be investigated	5
1.4	Method of evaluation	5
1.5	Available data	5
1.5.1	Data characteristics	6
1.5.2	Problems	7
1.6	Outline	9
2	Background	11
2.1	Previous projects	12
2.1.1	Visualisation of diagnosis histories: NSEPter	12
2.1.2	Depth project: Visualisation of collections of patient histories	13
2.2	Information visualisation	15
2.2.1	Preattentive processing	16
2.2.2	Choice of shapes	16
2.2.3	Choice of colours	18
2.3	Interactive computer graphics	21
2.3.1	Cost of knowledge	21
2.3.2	A cognitive interaction model	22
2.3.3	Interaction techniques	23
2.4	Related visualisations	25
2.4.1	Event charts	26
2.4.2	Lexis pencils	28
2.4.3	History visualisations	29
2.4.4	Visualisation of temporal queries and results	31

Contents

2.4.5	Comparison	33
2.5	Evaluation methods	34
2.5.1	Empirical evaluation	34
2.5.2	Expert reviews	36
2.5.3	Relevant taxonomies	38
2.5.4	Comparison	40
3	Proposed design	43
3.1	Data model	44
3.1.1	Conceptual hierarchies and regular expressions	44
3.1.2	Adoption of the available data	48
3.2	A static visualisation of patient histories	50
3.2.1	Events	52
3.2.2	Intervals	53
3.2.3	Axes	54
3.3	Adding interaction	54
3.3.1	Selecting information to visualise	56
3.3.2	Information availability: Details-on-demand	56
3.3.3	Query-based operations	57
3.4	Temporal query language	60
3.4.1	Language definition	61
3.4.2	Algorithms	66
4	Case study	91
4.1	Goals	92
4.2	Performing the study	92
4.2.1	Limitations	93
4.3	Findings	94
4.3.1	Introductory session	94
4.3.2	Query 1: Starting medication	95
4.3.3	Query 2: Prescribing ACE-inhibitors	97
4.3.4	Query 3: Complications of hypertension	99
4.3.5	General observations	100
4.4	Summary of findings related to goals	101
5	Discussion	103
5.1	About the informal evaluation	104
5.1.1	Case study	104
5.1.2	Indications of clinical relevance	106
5.2	Relating the proposed design to other work	107

5.2.1	Heuristic evaluation	107
5.2.2	Relation to information visualisation taxonomies	111
5.2.3	Relation to state of the art	112
5.3	Suggested improvements	114
5.3.1	Visualisation	114
5.3.2	Interaction	119
5.3.3	Query language	127
6	Conclusion	135
6.1	Evaluation of our approach	136
6.2	Summary of contributions	137
6.3	Future work	137
A	Assignment text	139
B	Abbreviations and terms	141
C	Complete data model	143
D	Paper	145
E	Digital appendix	149
E.1	Running the prototype	149
E.2	Command language	150

List of Figures

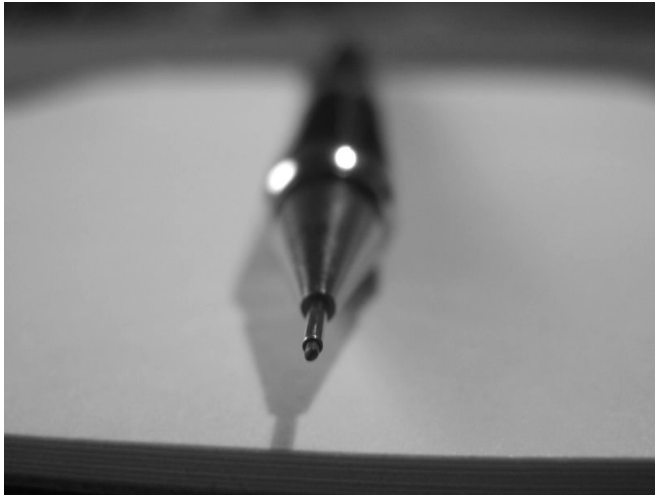
1.1	The visualisation prototype	4
1.2	Length of histories	7
2.1	NSEPter prototype	14
	(a) View of a merged graph	14
	(b) Zoomed-out view of a merged graph	14
2.2	Preattentive search	16
2.3	Visual grammar of diagrams	17
2.4	<i>CIEluv</i> chromaticity diagram	19
2.5	Basic event charts	27
	(a) Calendar event chart	27
	(b) Interval event chart	27
	(c) Goldman event chart	27
2.6	Extended event charts	28
	(a) Plotting by continuous covariate	28
	(b) Aligned event chart	28
2.7	3D lexis pencil diagram	29
2.8	2D lexis pencil diagram	30
2.9	LifeLines	31
2.10	Life course visualisation	32
2.11	Query result visualisation	33
2.12	Criteria for evaluating information visualisations	37
2.13	Criteria for evaluating interaction mechanisms	38
3.1	Data model	45
3.2	Visual representations	51
3.3	The prototype user interface	55

List of Figures

	(a) The prototype main window	55
	(b) The information pane	55
	(c) The colour legend	55
3.4	Filtering	57
	(a) No filtering	57
	(b) Filtered events	57
	(c) Filtered histories	57
3.5	Health record viewer	58
3.6	Alignment	59
	(a) Unaligned histories	59
	(b) Aligned on first hypertension	59
3.7	Sorting	59
3.8	Tokenizing and parsing a query	61
3.9	Lazy execution of a query	68
5.1	Distribution of cases	106
5.2	Object displays	118
	(a) Chernoff faces	118
	(b) Star glyph	118
	(c) Another star glyph (radar plot)	118
C.1	Data model with all entities	143

List of Algorithms

1	Extraction of all matches	72
2	Event search algorithm	73
3	First-query algorithm	73
4	Disjunction query algorithm	75
5	Window-with query algorithm	76
6	Merge query algorithm	78
7	Caching proxy for ExecuteQuery	80
8	Sequence query algorithm	82
9	Conjunction query algorithm	84
10	Window-without query algorithm	86



Introduction

(chapter one)

Most current tools for inspection of treatment histories focus on presenting information about the individual patient. We believe that interesting knowledge can be discovered by investigating *groups* of patients, and that information visualisation is well suited for this task. Hence, our research question¹ is as follows:

¹Assignment in appendix A.

How can we use information visualisation to support retrospective, explorative analysis of collections of patient histories?

1.1 Motivation

Information visualisation enables visual inspection of the data in a way that “*forces* us to notice **what we never expected to see**”²; the intuition is the same as behind statements as “I see what you mean”. Stuart Card gives a definition:

²Quote from [MM00], quoting Playfair’s political atlas from 1786; the emphases are from Playfair’s original.

Information visualisation (. . .) is the use of interactive visual representations of abstract data to amplify cognition. [War04, Foreword]

This amplification serves as an extension of our limited short-term memory, allowing a complex task to be approached by analysing visual patterns. Also, a visualisation may reveal artifacts in the data: In one instance, statisticians were analysing a data set for a long time³ before one of them realised that they had mixed up part of the data – a fact that was discovered using a visualisation [MMoo]. From this example, it seems that visualisations have the ability to generate insight.

³According to [MMoo], the data set was included in “a number of statistics texts”.

One aspect of applying such techniques to a new domain is that of choosing appropriate graphical representations. This is not entirely straight-forward, as illustrated by an example [War04]: Traditionally, nautical charts with contour lines are used to depict echo sounder scanning results (scanning of the sea-floor). However, when one such data set was plotted in a different manner, as a height field⁴, new patterns emerged: First, the degree of noise induced by the rolling of the ship performing the measurements was revealed, indicating that the models used for compensation were insufficient. Furthermore, the new visualisation displayed surprising geological characteristics, leading to new insight and a scientific publication. Hence, the choice of graphical arrangement is vital to the opportunities for gaining insight.

⁴Drawing a 3D “terrain” where the altitude of a point corresponds to a data value.

The above definition adds a second dimension: Interaction. Extending the static diagram with interactive operations allows the analyst to see different views of the same data, helping to form a more complete understanding. Comparisons can be composed, bringing different parts of the chart together, and the level of detail can be varied dynamically. This allows the analyst to build up a mental model of the data, using the visualisation as an extension of his short-time memory.

We believe that the application of information visualisation techniques to collections of patient histories may be a valuable addition to the current repository of Electronic Health Record (EHR) tools: Current EHR tools constitute a computerisation of the traditional health record, allowing the user to view and manipulate the individual record on the screen, mainly in text form. Projects such as the *LifeLines* project [PHL⁺98, SSP⁺98, PMR⁺95] aim to enhance usability of the EHR through information visualisation. However, the emphasis is still on the individual patient. This thesis addresses the issue of designing an information visualisation with accompanying explorative tools, facilitating retrospective investigation of a *collection* of patient histories.

1.2 Our approach to the problem

Analysing the question posed at the very beginning of this chapter, we refine the problem statement to the following questions:

- How can suitable information visualisation techniques be applied? This includes both user interface design and usability considerations.
- What are the interesting properties of patient histories, and how can meaningful groups of these be extracted? To answer this question, a relevant set of operations must be devised, and efficient algorithms must be proposed and analysed. Also, database-technical issues may have to be considered.
- How can the visual representations of the different elements in the histories be designed to harness the capabilities of the visual processing system of the user? This question is related to cognitive psychology.

Each of these questions can be approached separately. However, since we are neither physicians nor psychologists, we do not possess the specific knowledge needed to perform cognitive experiments or to define what constitutes “meaningful groups”. Having previously performed two projects working towards the same goal as this thesis ([ØHN05, Nor05]; see section 2.1), we have experienced the problems of gathering requirements from our potential users: They are not aware of their needs, and of how information visualisation techniques can be applied to enable them to view data in a way that was not possible before.

In one of the earlier projects we built a prototype, and when we demonstrated that to a physician, he provided valuable comments on the design. Motivated by this, and by the ideas and insight we earned through the previous projects, we decided to build a prototype system. This implies that we must approach all questions at once, and that each question cannot be treated with as much detail as if they were addressed individually. On the other hand, we gain the opportunity to apply our experience from the previous projects and obtain feedback from real users on a concrete idea.

The feedback will be generated from a test of the prototype: From this test, we hope that we will gain more insight into the problem, and that we will be able to discuss the usefulness of our design in order to improve it. Thus, our process consists of three stages:

- With basis in the experiences from our previous work, design and implement

a visualisation prototype. This part is supported by studying relevant literature and research.

- Perform a preliminary case study with a potential user to obtain feedback on the design.
- Discuss the results of the case study as basis for improvement. This part is also supported by the literature review.

This thesis has been written in parallel with the development process, and as a consequence, it shows how the case study and our analysis of the prototype has resulted in concrete suggestions for improvement.

Figure 1.1 shows the main window of the above-mentioned prototype, along with a brief explanation of the elements in it (see chapter 3 for a more in-depth explanation).

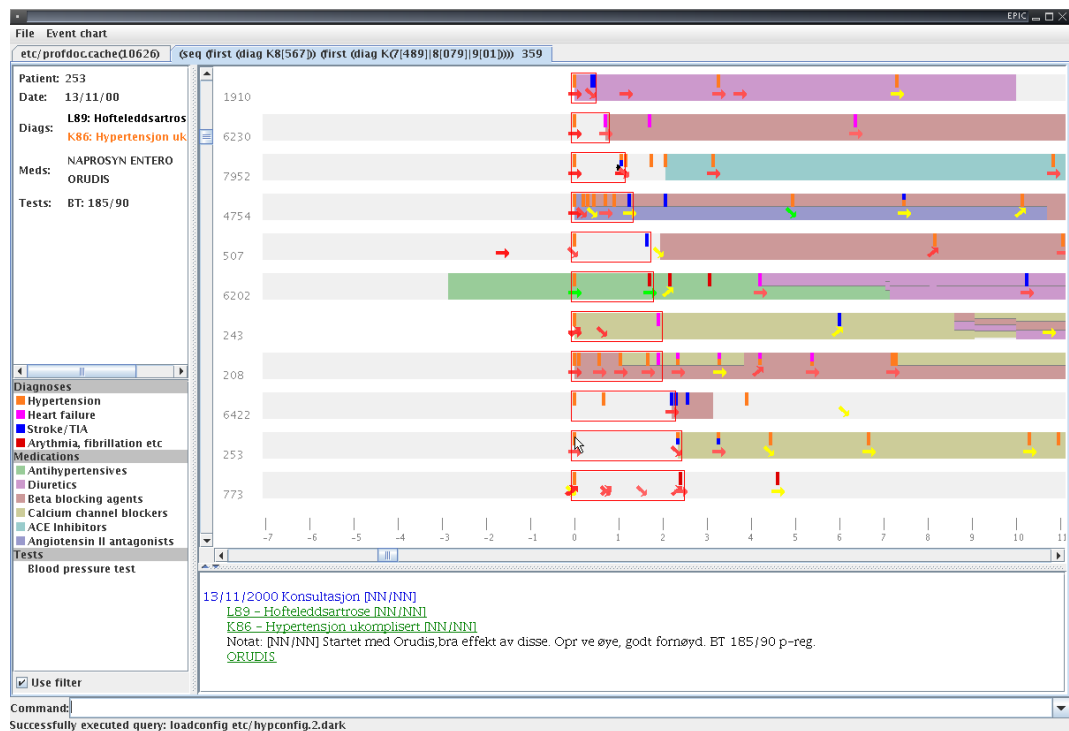


Figure 1.1: The visualisation prototype (copy of figure 3.3(a)). Each gray bar in this figure constitutes a patient history, with small rectangles and arrows indicating diagnoses and blood pressure measurements, respectively. The colours in the background show different classes of medication. On the left-hand side and bottom of the window, there are dynamic displays showing detailed information about the history content under the mouse cursor.

1.3 Case to be investigated

To constrain the problem, we choose to concentrate on hypertension patients in primary care. When we consulted a general practitioner on which problem area to focus on, we were recommended to study hypertension, because:

- There are many medical guidelines that apply to this problem, making it possible to compare the data to recommended practice.
- The data we have available for research, which is a database from a general practitioner's health centre (see section 1.5), contains a relatively large number of hypertension patients, since this is a common diagnosis. Moreover, these histories are typically long since the problem is chronic and most of the treatment happens at the primary doctor's office.
- The question of when to start medical treatment of hypertension is somewhat controversial, concerning both the actual conditions for when to begin, and the pharmaceutical industry's role. This means that there could be interesting facts to be discovered about this group of patients.

1.4 Method of evaluation

Since little is known about our design's utility at the start of the project, we choose to develop an interactive prototype and informally test it through a preliminary case study. From the results of this study it ought to be possible to decide if further research is warranted, and what paths of development seem the most promising. In this study, we let a general practitioner examine known data (from his own health centre), looking for both new and well-known information. From his initial comments, we formulated three cases that we investigated more closely.

1.5 Available data

At the Norwegian Centre of Electronic Health Records Research (NSEP), a relational database of Electronic Health Records for 10,515 patients is available. The records contain contact diagnoses, prescriptions, notes taken at the consultations, sick-leaves, and more. In the visualisation prototype, the following types of data are handled:

- *Contacts*: There are several ways in which the patient and physician maintain contact, such as consultations, letters or by telephone. Each contact is typically associated with contact diagnoses and possibly prescriptions and test results. In addition, the physician makes notes about the contact. These are vital for the understanding, but not suited for direct visualisation. Some information (blood pressure measurements; see *tests*, below) can be extracted from these text blocks.
- *Diagnoses*: Reasons for contact, the patient's concerns and complaints, and illness diagnoses are coded in the International Classification of Primary Care (ICPC) [WON98].
- *Tests*: This includes a wide range of measurements, such as blood pressure measurements, allergy testing, and urine samples. Test results often have one or more scalar or categorical values, or the results can be categorised (i.e. normal or high blood pressure). In the case of several tests of the same type appearing in succession, it is relevant to discuss the trend as well. In the prototype, only blood pressure measurements are visualised.
- *Prescriptions/medications*: Drugs are prescribed by the physician, and each prescription is recorded in the database, coded in the Anatomical Therapeutic Chemical Classification System (ATC) [WHO05]. A drug is meant to be taken for a length of time, and this time interval (medication interval) can be guessed from the data found in the prescription record. Around the time a medication interval expires, the doctor may choose to renew the prescription, and this is entered in the database as a separate prescription entry. From the sequence of medication intervals for a given drug, the time interval the patient is treated with that drug can be deduced.

1.5.1 Data characteristics

Part of the database is structured, i.e. the data is available in given fields and coded in a standard way. This is the case for diagnoses and prescriptions. However, some of the information related to prescriptions is missing or invalid, and it is necessary to repair some of the data (see section 3.1.2). Other useful data items, such as blood pressure measurements, are represented as part of the free-text in the contact notes. We extract this data using regular expressions (see section 3.1.1). Since data coded in free text is subject to typing errors and different ways of representing the information, this extraction is likely to be incomplete (see section 1.5.2 for examples).

Figure 1.2 shows a plot of history length versus the number of histories with this length, for 95% of the histories. It can be seen that most histories are relatively short.

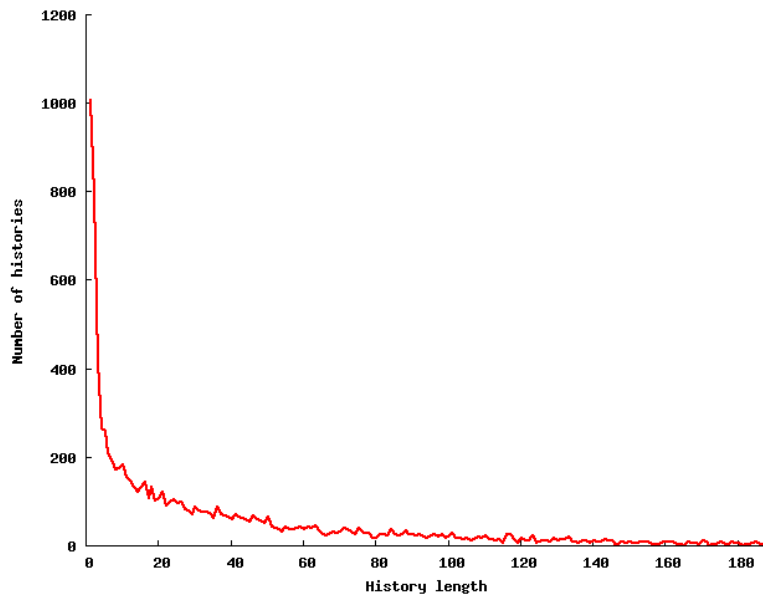


Figure 1.2: Plot of the number of histories with a given length versus history length. The plot shows the 95% shortest histories.

1.5.2 Problems

The available data poses at least two problems: Legal issues related to privacy concerns in a personal data filing system, and issues related to errors and noise in the data.

PRIVACY CONCERNS

An Electronic Health Record (EHR) is a private document, and handling of EHRs is regulated by the Personal Data Act (Personopplysningsloven [Jop01]), the Health Personnel Act (Helsepersonelloven [H0001]), and the Health Data Filing System Act (Helseregisterloven [H0002]). This implies that both information security and privacy issues must be considered when handling the data. To diminish such requirements, the data we are using has been *anonymised* and in the legal sense *released*. In this process, identifying properties of the data, such as names, places, dates and

diagnoses of low frequency have been removed, and the person responsible for the data has approved that we are free to use it and publish our results.

ERRORS AND NOISE

EHRs are entered by hand, and this can be an error-prone process. Examples of this include use of non-existent diagnosis codes, missing codification of diagnoses and medications, and wrong dates (such as patients seeing their doctor several decades prior to their birth). In addition, part of the database was damaged during an upgrade. There are several sources of noise in the data, as explained below.

- Different explanations People see the doctor for a variety of reasons, and the entries related to the problem one is studying may very well be interleaved by totally unrelated ones. This can be partially remedied by filtering out items that are not thought to be relevant, but static filters cannot handle all cases. For example: A patient that is treated for hypertension falls and injures a knee. For this injury, a diuretic drug is prescribed – a drug which is also commonly used in the treatment of hypertension. In this case, domain knowledge and inspection of the contact note for the prescription of the drug is needed to decide that the diuretic is not related to the patient’s hypertension.
- Free-text Some vital information is entered as part of free-text fields, such as blood pressure measurements and dosages for drugs. This involves parsing of text with typing errors and differing conventions for entering values. For instance, blood pressure measurements appear as part of the contact note, most often on the form “BT: <integer>/<integer>”. However, there are also many instances of constructs like “measured the BT to be 195/100”, or even “BT was 195-210/100”, where the latter indicates that several measurements were performed. It is difficult to ensure that all text fields have been parsed correctly (and usually way too many instances to read through manually for control).
- Missing data Data can be missing from the structured fields, appearing in the free-text fields only. This is true for many prescriptions, where only the name of the drug is given, not the code describing it. We remedy this by looking for other prescriptions with the same drug name and an ATC code (see section 3.1.2).
- Varying motivations The system of paying for health services in Norway is organised so that the clinic gets paid based on which diagnoses have been given. This means that diagnosis codes probably are chosen not only to describe the state of the patient, but for mercantile reasons also – especially if different codes may be used to describe the problem.

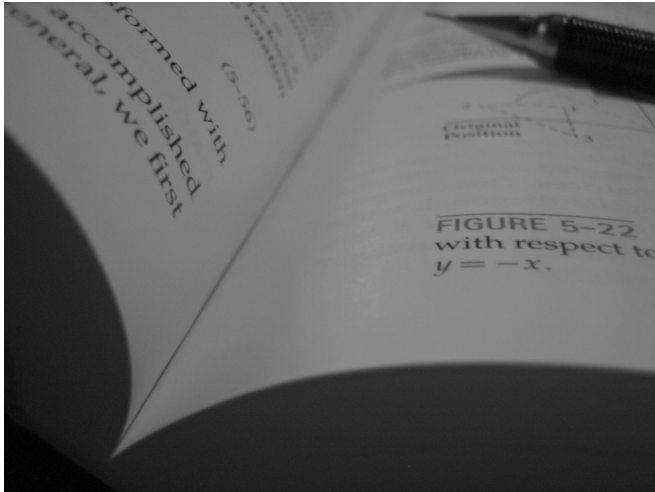
1.6 Outline

The rest of this thesis is organised as follows:

- *Background:* Previous projects that we have undertaken are quickly summarised. Furthermore, this chapter reviews material on how humans perceive visual structures, and how this influences on the choice of shapes and colours in information visualisations. Also, interactive operations are introduced in a context of cognitive considerations.
As a basis for the discussion of our design, we describe related work and existing solutions. Also, we review different guidelines for information visualisation and evaluation techniques.
- *Proposed design:* We describe a design for presenting collections of histories, applied in the medical domain through an interactive prototype. This chapter introduces the data model we use, and the techniques we apply to repair the data. A static visualisation is described, and interactive features are added to it. We perform some of the operations on both diagram and data using a temporal query language. This is described in the last section, along with a formal definition of its operators and a set of algorithms implementing them.
- *Case study:* To evaluate our design, we performed a preliminary case study to prepare the grounds for a more thorough evaluation later. In this study, a general practitioner used our prototype to investigate a collection of hypertension patients and comment on his findings. Three different cases were studied.
- *Discussion:* We discuss our design based on the experience from the case study, and on theory and research. Furthermore, we position our solution in relation to other work in the field. Finally, we suggest how the design may be improved, and we introduce alternative solutions to selected problems.
- *Conclusion:* In the final chapter, we conclude that our design seems promising enough to warrant the commission of a research project during the summer and autumn of 2006 to further investigate its feasibility.
- *Appendices*
 - A: The original assignment text.
 - B: Abbreviations and terms.
 - C: A diagram of the complete model of the data being represented in the system (as opposed to the partial model of what is being visualised shown in section 3.1).
 - D: A paper that was submitted to the IDAMAP-2006 workshop.
 - E: Description of the digital appendix, with a brief listing of the features of the command language in the prototype.

Chapter 1. Introduction

Throughout the thesis, definitions and detailed information will be shown in boxes with grey background. Certain illustrations contain data from a real Electronic Health Record (EHR); these have been approved for printing, in the sense that they do not contain identifying information, by the responsible physician (Anders Grimsmo).



Background

(chapter two)

This chapter introduces our former work and describes theory and research related to the interactive visualisation design proposed in this thesis. It treats the following subjects:

1. *Previous projects*: During the summer and autumn of 2005, two projects aiming to visualise collections of patient histories were performed. These projects form the basis on which this work is founded.
2. *Information visualisation*: Using graphics to convey information or knowledge and facilitate the possible discovery of unknown properties of the data. This section discusses the principles of preattentive processing and gives guidelines for selecting shapes and colours for a visual representation.
3. *Interactive computer graphics*: Concerning how user interaction can be designed to allow effective navigation of an information visualisation. This section introduces models for interaction founded in cognitive psychology and then describes actual interaction techniques.
4. *Related visualisations*: Visualisation designs and tools that resemble or relate to the design to be described in chapter 3.
5. *Evaluation methods*: Taxonomies and techniques for assessing the usefulness of an information visualisation for a particular purpose.

2.1 Previous projects

Prior to this thesis, we had performed two projects: During the summer of 2005, a prototype visualising diagnosis histories was built, and during the autumn of the same year, we investigated several other designs. This section summarises the prior work and lists the most important results and experiences.

2.1.1 Visualisation of diagnosis histories: NSEPter

Two developers worked for eight weeks on the project named *NSEPter*, a visualisation prototype using directed graphs to portray collections of patient histories. The only information from the EHR that was utilised, was the diagnosis codes for each patient.

NSEPter had the following functionality [ØHN05]:

- Each history was laid out on a horizontal line, and each diagnosis code was represented by a node, with an edge between nodes representing diagnoses adjacent to each other in the history.
- The system was capable of searching for diagnosis instances based on a regular expression over ICPC codes. This search could be used to hide or show individual nodes, or it could operate on the level of histories, based on the presence or absence of a given code.
- Nodes could be *merged*: The users specified a regular expression over the ICPC codes, and the application merged nodes with codes matching the given expression into one. This was performed serially from the beginning of the histories, so that the first occurrence of a node from one history was merged with the first from all the other histories, the second was merged with the second, and so on. From each merged node, the process could be recursively applied to neighbouring nodes in both directions, in a hope that the histories would exhibit similar patterns before or after an important event. Common edges between merged nodes were scaled according to the number of histories exhibiting the transition in question.
- NSEPter had a plug-in architecture in which filters and visualisation engines could be interchanged, all operating on the same data model.

Figure 2.1(a) shows the NSEPter prototype in action, showing a graph of diabetes patients. Here, the thicker lines indicate that several patients follow the same path before and after the diabetes code, T90, which in this case is the first occurrence in all the histories.

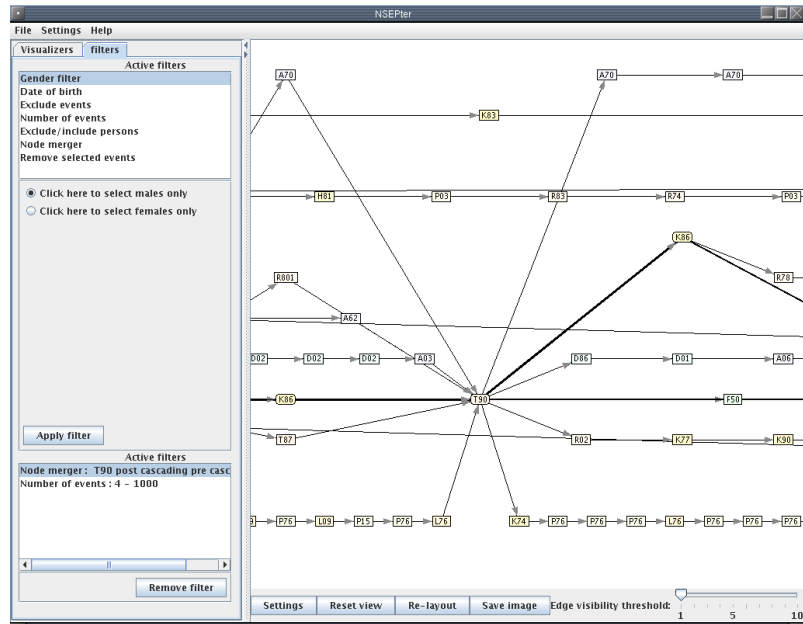
This prototype had several weaknesses:

- NSEPter's main problem was the transformation from histories to graph representation: The dimension of time was lost. This made medically vital information absent in the visualisation, and it was no way of deciding if the time that passed between two events was half an hour or a year.
- The graphs quickly became crowded and virtually unreadable, and they used a lot of screen space. If one zoomed out to see the big picture, the visualisation was basically a web of edges, and with larger zoom factors, context was lost, and it was difficult to determine what one was looking at. Figure 2.1(b) illustrates this.
- Our merging algorithm was not very noise-resilient. It would miss an opportunity to merge nodes if two histories differed in one single position. Moreover, the order in which the histories were merged, mattered.

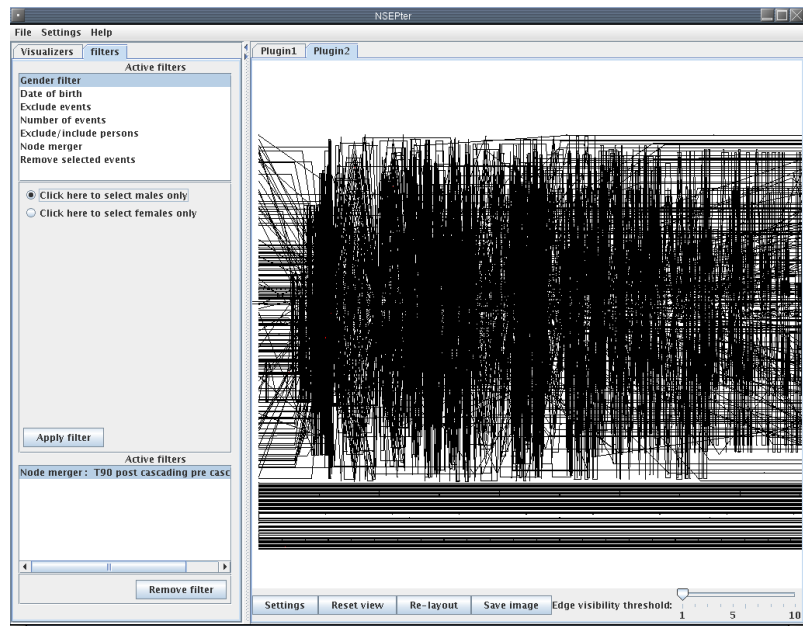
2.1.2 Depth project: Visualisation of collections of patient histories

During the autumn of 2005, a depth project was performed [Nor05], focussing on improving the basic idea of using directed graphs and merging similar paths. This project was mainly concerned with processing: Heuristics for multiple alignment were employed, and different measures to reduce the amount of noise were attempted. Also, we calculated abstractions over sequences of diagnosis instances and mined for relations between the diagnosis codes themselves.

From this project, we gained experience in processing the available data. In particular, the filtering and selection techniques presented in this thesis build on this experience, but it could be interesting to include the abstractions as well at a later stage. Among the future work listed in the conclusion of [Nor05], were goals such as to incorporate more information in the visualisation and to a greater degree exploit the dimension of time.



(a) View of a merged graph



(b) Zoomed-out view of a merged graph

Figure 2.1: The NSEPter prototype showing (a) a small graph, merged around the first incidence of diabetes, (b) several hundred patients, showing the entire graph.

2.2 Information visualisation

The human brain possesses a considerable capacity of pattern recognition through visual processing, while the ability for processing large amounts of text or numbers is limited in comparison. For microcomputers, the situation is opposite: While possessing the ability for processing huge amounts of data, the algorithms for pattern-recognition are limited, confined to finding one or a few types of solutions, and often they are sensitive to noise and errors. Other benefits of human cognitive abilities are access to creativity and the availability of domain knowledge, conscious or not, stored in the heads of users [TSK05].

Our disposition to learn and understand through vision is an important motivation for the development of the discipline of Information Visualisation, where the data is presented in a visual form. One of the earliest uses of graphics for depicting statistical data is found in a drawing from 1644 [Tuf97]. Using the computer for number crunching and construction of a visual representation, the strength of man and machine are combined for extraction of interesting properties of the data being investigated. The high availability of computing power and high-resolution displays makes information visualisation attractive and feasible, even on a low-end home computer.

When designing an information visualisation, knowledge of how different visual structures are perceived serves as a useful guideline in making the visualisation easy to learn and understand. This includes choosing graphical primitives that can be quickly identified, and arranging them for the best possible exploitation of the perceptual system. If the visualisation is well crafted, searching for specific information becomes easy (can be done preattentively, see below), and patterns are revealed.

In addition, cognitive limitations must be observed: When an image becomes too complex, it becomes difficult to read and interpret. Also, the short-term memory for graphical elements is quite limited. In fact, experiments indicate that we are not able to detect even large differences when a display changes abruptly [SA05].

Throughout this section, the following topics are treated:

1. *Preattentive processing*: A brief introduction to the most efficient cognitive processing technique, preattentive processing, where elements are located without consciously looking for them.
2. *Choice of shapes*: Suggestions on how different shapes are perceived, to form a

basis for discussing the visual representations to be proposed in chapter 3.

3. *Choice of colours*: Relevant theory on the perception of colours, acting as a guideline in choosing colours that contribute to the clarity of the communicated information.

2.2.1 Preattentive processing

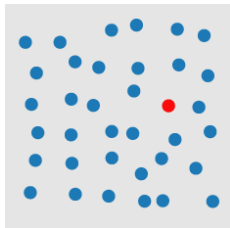


Figure 2.2: Find the red circle [Hea99]

Sometimes, features seem to “pop out” of an image. For example, when trying to locate the red circle among the blue ones in figure 2.2, the differing colour is picked out by the visual processing system before the signal reaches the centre of attention – it is performed *preattentively*. The time used to process the visualisation (search for the red circle) is independent of the number of distracting elements (blue circles). There are a number of features that can be processed preattentively; for example is the same effect evident when searching for circles in a figure with many squares or other angled forms [Hea99].

On the other hand: Searching for a red circle in a figure with many blue circles and red squares cannot be performed preattentively, and the time to do so increases linearly with the number of distracting elements [Hea99, Waro4]. This is called a *conjunction search*, since two properties need to be processed in order to identify the target: It has to be red, and it has to be circular. In general, conjunction search is not preattentive, but there are important exceptions [Waro4].

Since preattentive processing is much faster than its counterpart, choosing a suitable visual encoding is important for the efficiency of the resulting presentation. This includes choosing good colours and distinct forms, and avoiding the need for conjunction search.

2.2.2 Choice of shapes

Figure 2.3 shows a table from [Waro4] of the *visual grammar of diagrams*: A set of abstractions common to node-link diagrams. Even though our design does not include a node-link diagram, several of the properties (concerning containedness, distinctness and spatial organisation) will be relevant as an aid in assessing the cognitive quality of the chosen representation.










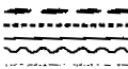



Graphical Code	Visual Instantiation	Semantics
1. Closed contour.		Entity, object, node.
2. Shape of closed region.		Entity type.
3. Color of enclosed region.		Entity type.
4. Size of enclosed region.		Entity value. Larger = more.
5. Partitioning lines within enclosed region.		Entity partitions are created, e.g., TreeMaps.
6. Attached shapes.		Attached entities. Part-of relations.
7. Shapes enclosed by contour.		Contained entities.
8. Spatially ordered shapes.		A sequence.
9. Linking line.		Relationship between entities.
10. Linking-line quality.		Type of relationship between entities.
11. Linking-line thickness.		Strength of relationship between entities.
12. Tab connector.		A fit between components.
13. Proximity.		Groups of components.

Figure 2.3: The visual grammar of diagrams, copied from [War04, Figure 6.33].

In addition, preattentive concerns should be addressed when selecting shapes. Shapes that are preattentively processed are often simple shapes, and the different shapes that are chosen should be sufficiently distinct. Ware lists different features that are preattentively processed (in verbatim from [War04]):

- Line orientation
- Line length
- Line width
- Line collinearity
- Size
- Curvature
- Spatial grouping
- Blur
- Added marks
- Numerosity
- Colour hue
- Colour intensity
- Flicker
- Direction of motion
- 2D position
- Stereoscopic depth
- Convex/concave shape from shading

2.2.3 Choice of colours

Edward Tufte describes the problem of choosing colours for information visualisations in his book “Envisioning information”:

“(...)even putting a good color in a good place is a complex matter. Indeed so difficult and subtle that avoiding catastrophe becomes the first principle in bringing color to information: Above all, do no harm.” [Tuf90]

He lists four uses for colour in information design:

- *Colour as noun*: Labelling different sorts of information.
- *Colour as quantity*: Encoding a quantity with colour.
- *Colour as representation*: Representing or imitating reality with colours, for example representing water with the colour blue in a map.
- *Colour as beauty*: Using coloured decorations.

It is the first use of colour which is most relevant in this work, the second will also be touched. Concerning the use of colour for labelling data, Healey [Hea96] conducted an experiment concluding that preattentively distinguishable colours conform to the following three properties:

- *Colour distance*: The cartesian distance between any pair of colours in a perceptually balanced colour model, such as *CIE_{luv}*, should be sufficiently large. *CIE_{luv}* is a system devised by the International Commission on Illumination – Commission Internationale de l’Eclairage (CIE) – that codes colours with three components: L^* (luminosity) and (u^*, v^*) (together defining chromaticity)

[Hea96]. (u^*, v^*) are defined in terms of the luminosity L^* so that distances decrease with decreasing luminosity. This dependence on luminosity encodes the fact that dark colours are more difficult to distinguish – in the limiting case everything is black [War04]. Figure 2.4 shows the colours of this colour space for one value of L^* .

- *Linear separation*: The shortest euclidean distance from a colour to a straight line in colour space separating it from all other colours used in the visualisation should also be sufficient [Hea96]. Ware states this property by demanding every colour to lie outside the convex hull of all the others in colour space [War04].
- *Colour category*: Experiments show that it is possible to separate regions in colour space within which colours are consistently named. Furthermore, this subdivision seems to be culturally universal [War04]. Healey’s experiment indicates that also colour category must be taken into account when selecting colours that should be preattentively identified together [Hea96].

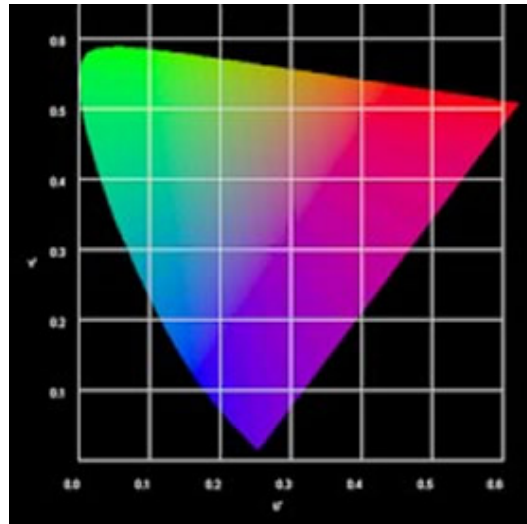


Figure 2.4: CIEluv chromaticity diagram

In addition, there are some other useful guidelines:

- *Contrast with background*: The background on which a colour is displayed can alter the perceived colour. One way to reduce such effects is to make sure there is a significant difference in luminosity between foreground and background [Tuf90]. Another possibility is to use a white or black frame around the coloured object [War04].

- *Colour blindness*: One model for the perception of colour is the *opponent process model*. Here, colours are perceived along the axes of black-white, red-green and blue-yellow. Colour-blind people have problems discerning colours that differ along the red-green axis. To make the displays readable for colour-blinds, colours should rather vary along the blue-yellow axis. However, this limits the design opportunities, and the red-green difference is considered the most effective (for people that are not colour-blind) [War04].
- *Number*: Only between five and ten colour codes may be used simultaneously for effective information coding [War04].
Healey tries to determine this empirically by using sets of 3, 5, 7 and 9 colours with maximal colour distance and linear separation (ignoring colour category) [Hea96]. In the experiments, subjects search for the presence of a rectangle of a specific colour among a collection of differently coloured rectangles. Several interesting observations are made:
 - The response time is dependent on which colour is searched.
 - Searching among 7 or 9 different colours is significantly slower than searching among 3 or 5.
 - Most targets could be found preattentively, but with the sets with 7 and 9 colours, some colours triggered a serial (non-preattentive) search behaviour. This was attributed to the fact that colour categories had not been taken into account when choosing colours for the experiment.In general, the number of colours that are practically usable in a single visualisation if preattentive search is desired is quite limited. This can be mathematically explained by geometry in colour space: When the number of colours increases, their pairwise distances will have to decrease, and so will their linear separation value. Healey concludes: “It appears that seven isoluminant colours is the maximum we can display at one time, while still allowing rapid and accurate identification of colours.” [Hea96] In this conclusion, *isoluminant* is the word that will allow for more than seven colours (by choosing a second set with a different luminance), but it is not clear how this will interfere with the already present colours, since there will be smaller differences in chromaticity.
- *Conventions*: It is useful to consider conventions in colour coding (e.g. green means good), but it must be observed that these conventions are culturally dependent (e.g. green signifies death in China) [War04].

As a practical help in selecting colours, Ware [War04] lists 12 colours he recommends for use in coding that are considered distinct in the sense of colour categories. The first six should normally be used before introducing any of the last six:

- | | |
|-----------|------------|
| 1. Red | 7. Pink |
| 2. Green | 8. Cyan |
| 3. Yellow | 9. Gray |
| 4. Blue | 10. Orange |
| 5. Black | 11. Brown |
| 6. White | 12. Purple |

2.3 Interactive computer graphics

Information visualisation techniques do not only concern the construction of a static diagram, but also how to interact with the visualisation. Using these techniques, parameters can be varied interactively to achieve a view of the data highlighting the properties the analyst is looking for. In addition, the level of detail may be varied, or the full details may be presented in another view.

This section describes interaction techniques that will be relevant for the visualisation proposed in chapter 3, and it is organised as follows:

1. *Cost of knowledge*: The notion of *cost of knowledge* is introduced and discussed as a basis for choosing efficient interaction techniques.
2. *A cognitive interaction model*: A model for how humans interact with visualisations is referred.
3. *Interaction techniques*: Techniques supporting the cognitive interaction model are presented. These techniques will be used in presentation of the chosen design in chapter 3 and in the discussion of it in chapter 5.

2.3.1 Cost of knowledge

An important measure in designing an effective interaction scheme is the *cost of knowledge*: The amount of energy that must be invested to extract a certain amount of information. Pirolli and Card compare a human's search for information to an animal's foraging for food; both seeking to minimise the amount of energy required to cover their needs. It is noted that information often appears in chunks or clusters separated by long distances of uninteresting data, just like edible plants do in nature. Furthermore, as scents may direct an animal to the food source, there may be

hints in a visualisation directing navigation towards the more interesting information [PC95].

This leads to a design methodology of analysing and minimising the effort needed to extract knowledge from a visualisation system. The cost of knowledge is twofold: First, there is the actual effort of locating and extracting the information. Then, one also has to consider the cost of not being able to do something else during the information search [Waro4].

To minimise the cognitive effort, two aspects should be addressed:

1. Reducing the cost of analysing the visual representation: Improving the visualisation to better support pre-attentive processing, and increasing compliance with cognitive limitations.
2. Reducing the cost of navigating the visualisation by choosing efficient interaction techniques.

2.3.2 A cognitive interaction model

The process of working with an interactive visualisation can be described as three nested loops that are being performed by the user [Waro4]: The problem-solving loop, the exploration and navigation loop, and the data manipulation loop. On the highest level, the analyst forms hypotheses about the data and refines these through revision of the visualisation. This is supported by the exploration and navigation loop, where the visualisation is navigated and the user builds a mental “map” of the data that is presented. On the lowest level, data items are identified and selected using basic motoric actions (such as eye-hand coordination).

In order to support the flow of the user’s work, the system must be responsive enough to avoid interrupting the thinking process. Shneiderman states that response times for mouse and typing actions should be less than 0.1 second [Shn98]. While lower response times generally lead to higher user satisfaction, there is a danger that the fast pace will increase the user’s error rate. When it comes to low-cost information seeking, Ware lists estimated times for different navigation tasks. For example, following a hyperlink takes two seconds, while using brushing¹ takes two seconds to relocate the mouse to the point of interest, and then 250 ms for each successive query.

¹Also known as “mouse-over” functionality, where the mouse cursor is placed over a representation to trigger display of detailed or related information about that representation.

Another aspect of response time is related to what is known as *change blindness*: If the user blinks or changes focus, or if the screen briefly blanks, between two successive views, it is probable that the user will be unable to detect the difference between the views. Even when searching actively for a difference, this task is difficult [SA05]. This means that the visualisation should not presume that a user is able to detect changes between views without a way of highlighting the change, such as with animation.

2.3.3 Interaction techniques

Given the cognitive interaction model described above, it must be a goal to find techniques supporting the two inner loops: Exploration and navigation, and data manipulation. This calls for effective navigation techniques using simple manipulations to find the information of interest, always supporting the higher-order goal of finding, refining or investigating a hypothesis.

Ben Schneiderman describes what he calls the Visual Information Seeking Mantra: “Overview first, zoom and filter, then details-on-demand” [Shn96], to which he devotes ten repeating lines in his article – once for each project in which it was rediscovered. This indicates the need for a visualisation that serves different purposes at different stages in the user’s process of gaining understanding of the data: First, an overview needs to be presented, to give the user a clue about what to look for. Then, relevant information must be sorted out, and all details should be easily accessible when the user needs them. It is interesting to note Colin Ware’s [War04] comments to this approach: He suggests that the process is not as directed as Shneiderman claims it to be, but rather an iterative procedure where interesting features are spotted, the view is zoomed out to get an overview, and then zoomed back in again for inspection of the details. Ware concludes that no matter how the process turns out to be performed, it is highly important that the visualisation acts as an interface capable of performing these operations.

The following paragraphs describe the tasks listed in a taxonomy by Shneiderman (in [Shn96]²), but additions from other sources are used as supplements in the presentation. While the first four tasks (overview, zoom, filter, details-on-demand) are frequently found in prototypes, the three latter (relationships, history, extraction) are more seldom [Shn96], since they do not add to the capability of the visualisation itself, “only” the user interface. They are, however, important for the explorative aspects of interaction and should be remembered when developing a prototype.

²The taxonomy is described in section 2.5.3

OVERVIEW

Looking at a new data set, the user would often not know what to look for. A simplified view of the data could give clues of interesting parts or patterns that warrant further investigation.

ZOOM

Zooming is a magnifying process. Typically, this involves enlarging visual representation of items (geometrical zoom), but the term does also apply to controlling the level of detail (semantical zoom) [HMM00, LA94]. This represents the transition from an overview to a more detailed view of a subset of the data. However, this process will typically hide other parts of the data, and it might lead to confusion regarding which subset of the data is actually shown. To remedy this, *focus+context*-techniques try to combine the zoomed-in subset with a course-grained view of the surrounding contextual data [LA94].

FILTER

Filtering is the process of removing or hiding unwanted items, and it is applied to draw attention to the items of interest and to avoid crowding. This process can be dynamic, such as by direct-manipulation controls [Shn96], or it can be specified using a query language. One problem of the latter is that of designing good user interfaces for specifying complex queries without needing to learn the query language.

DETAILS-ON-DEMAND

Details-on-demand refer to presentation of more detail when this is needed, often by clicking an item with the mouse, or simply by holding the mouse over the item of interest. This gives the user an opportunity to see the full details that were abstracted away in the visualisation, or the exact value of a data point.

RELATE

Shneiderman points out the need for showing and exploiting relationships between user interface and visualisation components: Clicking one component in the visu-

alisation could trigger an update of a dynamic query control in the user interface [Shn96]. Ware also describes this concept, asserting that a visual representation should not only be a “blob of color” on the screen, but also act as an interactive device [War04]. This form of relation should not be confused with “showing relationships within the data” – which indeed is a goal of information visualisation, but not part of relation as an interactive technique.

HISTORY

Keeping a history of actions to allow undo and redo encourage exploration by diminishing the consequences of choosing the wrong action, giving the user confidence in that he easily could get back to the current state if an action did not lead to satisfying results. In addition, the sequence of steps taken could tell a user what the current visualisation is showing. This is also pointed out by Lidwell et al [LHB03], when addressing the more general concept of *forgiveness* in the system. A forgiving system makes the user feel more secure in that he is warned before critical or irreversible actions are performed, and that other actions easily can be un-done. Ware [War04] cites a study showing that users work faster when they know that their actions can be reversed if they accidentally make an error.

EXTRACT

When the user has explored the visualisation and attained an interesting result, it is important to have the opportunity to save the data being viewed or the parameters used to extract it. The properties of the visualisation, such as colour mappings, could also be saved. Another useful property would be extraction of an image file that could be viewed with any image viewer, put on a web page or included in an e-mail. We implemented such a feature in an earlier visualisation prototype [ØHN05] (built on a different concept), and it proved to be a very useful feature.

2.4 Related visualisations

This section describes approaches to event or history visualisation that resemble the design to be introduced in chapter 3.

2.4.1 Event charts

Event charts are used to visualise individual events for a number of histories. Doing this may reveal correlations among events, and if the chart is constructed with respect to a covariate (see below), correlations between the covariate and the events can become evident. One important property of event charts is that they show the actual events of the history – as opposed to aggregated data – to let the “raw” data speak for itself.

An event chart depicts histories by showing one line for each history and plotting events using various symbols along this line. The horizontal axis is typically a time axis, while the vertical axis index histories. Lee et al [LHDoo] describe the three basic types of event charts, shown in figure 2.5:

- (a) *Calendar event chart*: The x-axis represents calendar time, while the y-axis index histories. This gives a “raw” view of what happened.
- (b) *Interval event chart*: The x-axis represents time relative to some common event of the histories, while the y-axis index histories. Using this chart can give insight into the development before and after a given event, especially if the histories are aligned on this event.
- (c) *Goldman event chart*: The x-axis represents time relative to some common event of the histories, while the y-axis positions histories according to calendar time of the common event. This is thus a combination of the event charts described above, showing two aspects of the data simultaneously.

Several extensions of the basic event charts exist [LHDoo]:

- *Sorting*: Ordering the lines along the vertical axis can facilitate identification of patterns with respect to the ordering criterion.
- *Grouping*: Separating lines according to a categorical time-invariant attribute (and possibly sorting each group separately) can highlight differences between the groups.
- *Plotting by continuous covariate*: In this context, a covariate is a time-invariant continuous attribute of the histories. The lines are placed along the vertical axis according to the value of this attribute. Note that this is different from sorting and grouping in the sense that the lines are spaced according to the difference in covariate, adding value over a mere sort. However, this raises the issue of

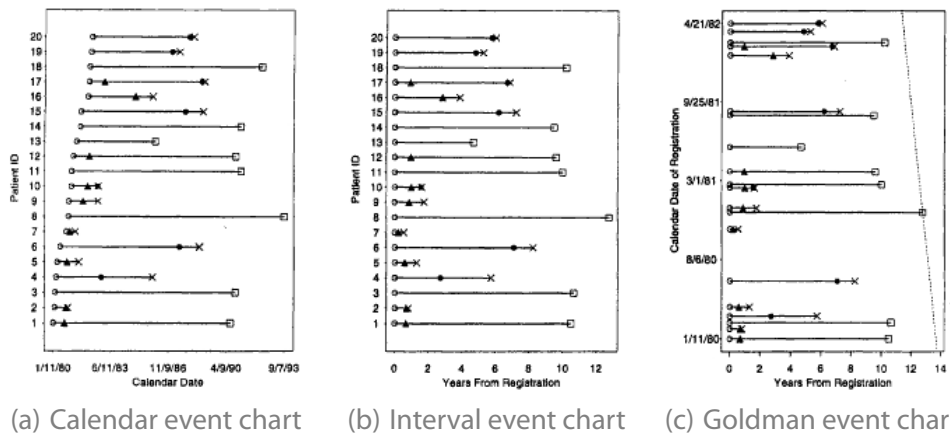


Figure 2.5: Examples of the three basic event chart types described by Lee et al, taken from [LHDoo]. The various symbols signify different events or classes of events.

how to handle the case where several lines have (nearly) the same value for the covariate. This could be solved by jittering (which would not be entirely correct).

The resulting charts look a bit like Goldman event charts; see the example in figure 2.6(a), plotted by age at the time of registration in a cancer study. In this plot, the lines seem to have been jittered around age 57 to avoid overdrawing. Note how the distribution of age is revealed, and how a few patients fall outside the main group.

- *Alignment:* Aligning the lines on a common event makes it possible to examine variation in development around that common event. See figure 2.6(b) for an example.
- *Changing line types:* Different line types can be used to show the value of a categorical covariate. In figure 2.6(b), solid lines signify that the patient is HIV-positive.

APPLICATIONS OF EVENT CHARTS

Lee et al [LHDoo] discusses the applicability of event charts as follow-up tools for clinical trials, concluding that “Compared with data printouts or tables, event charts provide a more efficient and more effective way of presenting data.” [LHDoo], giving a better overview. Furthermore, they state that event charts plotted against a covariate can be useful in building models and choosing proper analysis techniques. Also, the

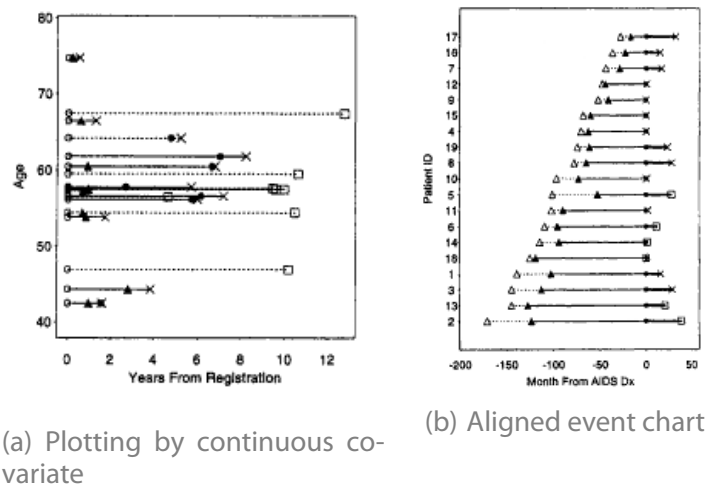


Figure 2.6: Examples of extended event charts, taken from [LHDoo]. (a) shows cancer patients’ history in an interval event chart plotted by age of the patient (the ‘x’ signifies death, the box is the last follow-up). (b) shows AIDS-patients aligned on the time they get the AIDS diagnosis (same symbols as (a); the filled-in triangle is the first positive HIV test).

charts give a good opportunity for error checking (detecting outliers).

Atherton et al [AJN⁺03] use event charts to summarise changes in quality-of-life data over time, and compare this technique to traditional scatter plots. The event charts used are interval event charts with different line styles and colours, and in some cases the lines are grouped according to a categorical attribute. They point out that the strength of this technique is the ability to clearly present temporal data, and conclude that application of event charts “has potential for use as a tracking device in oncology clinical trials.” [AJN⁺03]

2.4.2 Lexis pencils

Francis and Pritchard describe an event-chart like visualisation in , “Visualization of Event Histories” [FF96]. The metaphor in use is that of a pencil with multiple sides, with time running along the length of the pencil. Categorical variables are mapped to the sides of the pencil using coloured areas. These three-dimensional pencils are then placed in a diagram, ordered by two time-dependent variables in a three-dimensional coordinate system (see figure 2.7). As the authors point out, 3D diagrams pose sig-

nificant usability problems, and they propose a 2D variant shown in figure 2.8 in a later case study [FFP].

Several properties relate this visualisation to event charts, although not explicitly explained in the article:

- Lexis pencils show the raw data of the histories as marks on a linear representation, and several pencils are combined in a single diagram.
- Ranking and grouping of pencils are discussed, which is analogous to sorting and grouping in the extended event charts described in [LHDoo].
- Different choices for the variables on each axis are discussed, which can be seen as an extension of “plotting by continuous covariate”.

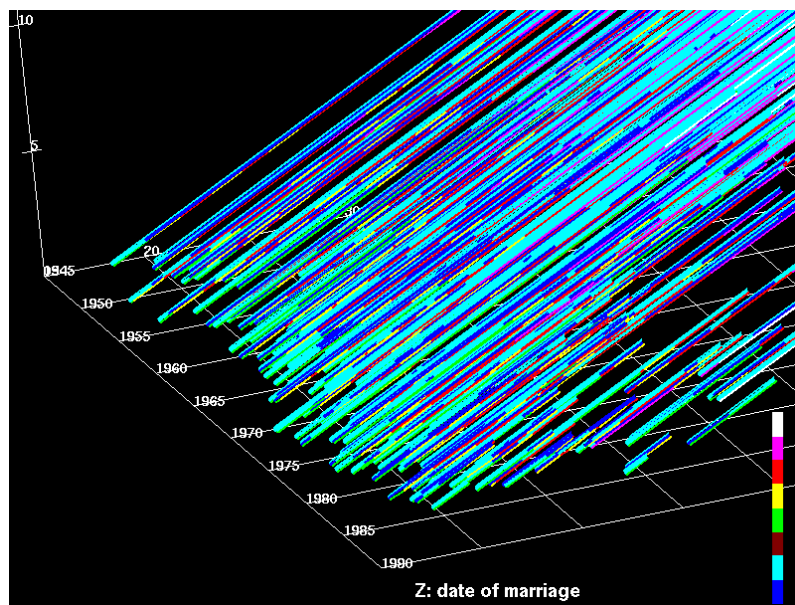


Figure 2.7: 3D lexis pencils showing employment histories of men and women in married couples in Kirkaldy along with the presence of children (as age of youngest child). These three variables are mapped to the three different faces of the “pencils” shown. Light blue means employed, while dark blue means unemployed. The diagram is from [FFP].

2.4.3 History visualisations

Visualisation of histories is an application of information visualisation that provides the user with an overview of data that often can be complex (life histories, health

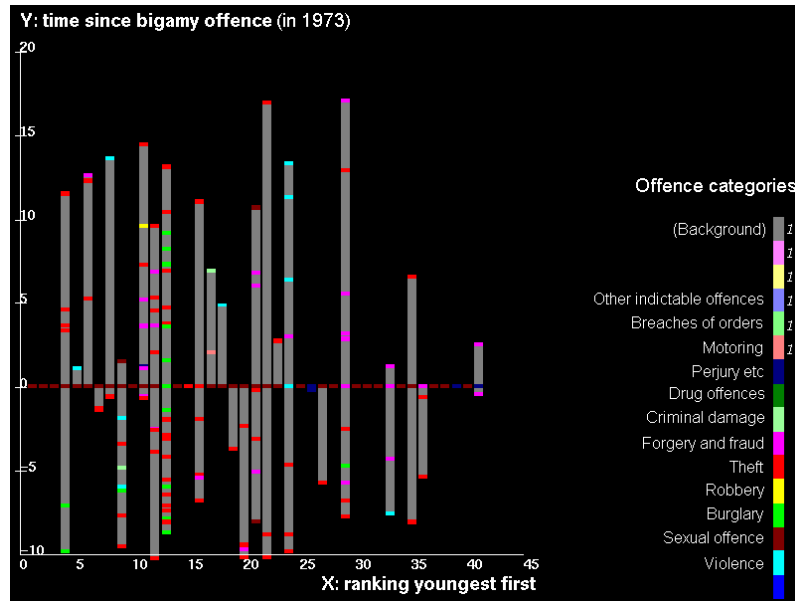


Figure 2.8: 2D lexis pencils showing bigamists’ criminal record aligned on the time of their conviction. Differently coloured markers are used to signify the different sorts of crime they get convicted for. The diagram is from [FFP].

records, etc). These views can also facilitate navigation of such data, and retrieval of the interesting parts from large amounts of data.

One such visualisation is *LifeLines* [PHL⁺98, SSP⁺98, PMR⁺95]: A LifeLine is an interactive timeline visualisation, plotting events for a single history grouped into different *facets*. Each facet contains information that is semantically related, such as (in a health record setting) diagnoses, medications, allergies, and general problems (e.g. smoker, depression). The visualisation can show information at different levels of abstraction: For example, medications can be shown using a name for the group of drugs (beta blocker) or by the individual drug names (atenolol, propranolol).

Figure 2.9 shows the *LifeLines* prototype running in a web browser. The upper part of the main panel shows the current patient with name and photo, in addition to a detail area showing detailed information about the elements under the mouse cursor. Most of the screen space is used for the LifeLine, which is subdivided into collapsible facets (Notes, Hosps, Tests, etc). The bottom panel possesses control functionality. To the right of the visualisation, there is an area for opening detail views such as X-ray images. The prototype supports zooming, and searching for related items (i.e. searching for “migraine” highlights all diagnoses and drugs related to migraine). In

addition, attributes can be mapped to different graphical representations by the user.

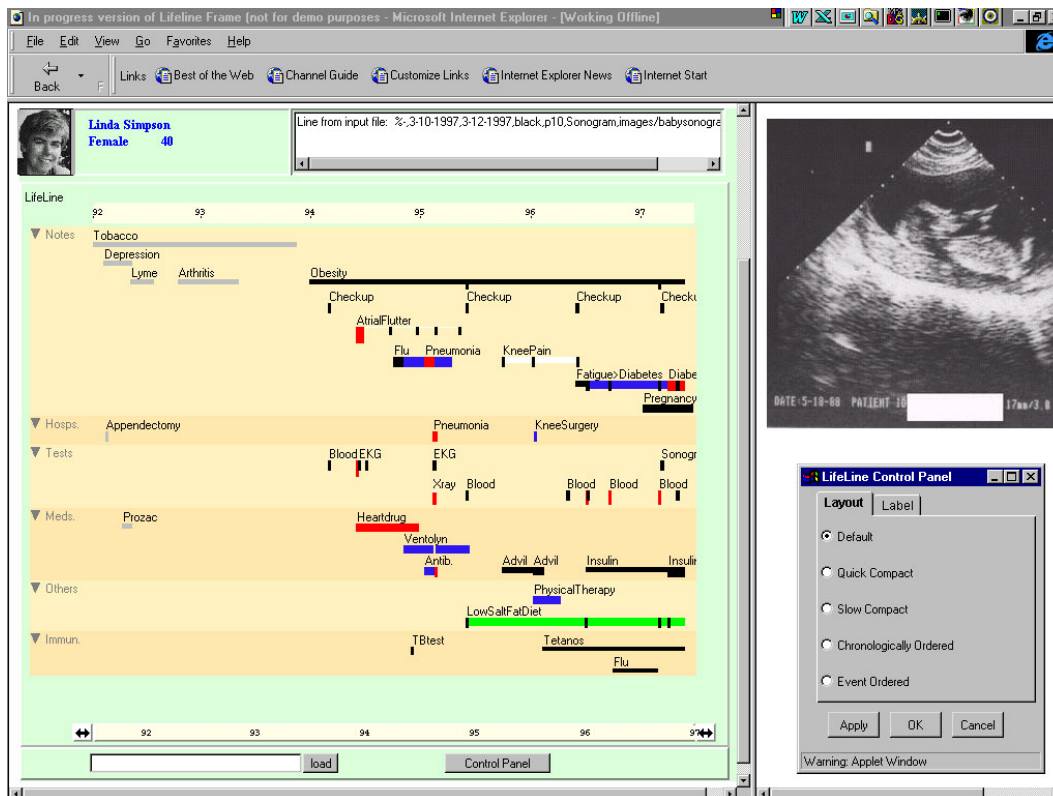


Figure 2.9: Screenshot of the LifeLines application retrieved from the project’s website, <http://www.cs.umd.edu/hcil/>

Maltz and Mullany [MMoo] cite an unpublished work by Klosak from 1999 and print a visualisation of a life course from it. This is given in figure 2.10. Here, different icons are used to signify the different events in the life of probationer “Megan”, and the shade of the icons, bright or dark, indicates if the event was positive or negative.

2.4.4 Visualisation of temporal queries and results

Another direction that is relevant to our work, is temporal queries and their results. Chittaro and Combi [CCo1] describe several metaphors for describing intervals with uncertain length: An elastic band, a spring, or a strip of paint. Representations of physical objects constrain the length of these representations to appeal to the user’s real-world experience. Their studies include usability tests of the different representations.

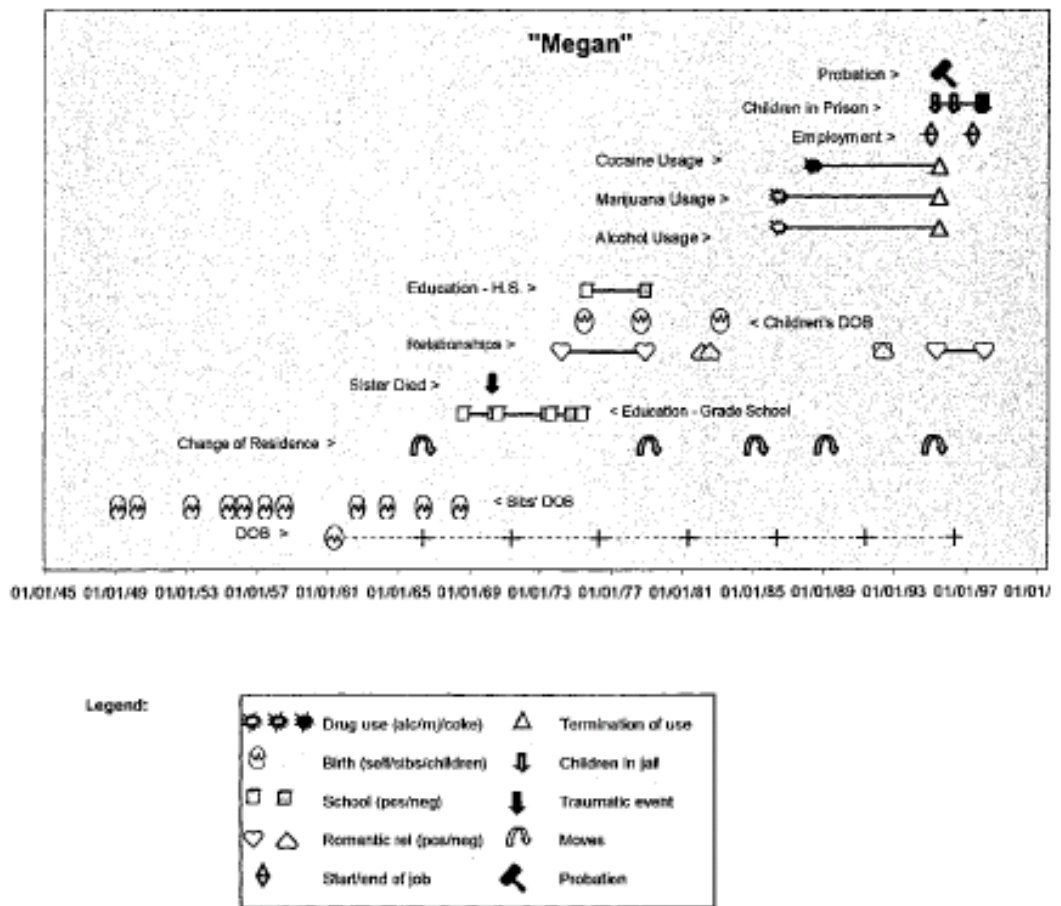


Figure 2.10: Visualisation of the life course of a female probationer, from [MMoo].

A recent³ project at the University of Maryland describes a user interface for the specification of temporal queries and a visualisation of the query results. In the report, the visualisation is described as a “graphical table of pattern matches” [FKSSo6]. This table (see figure 2.11) reminds of an event chart visualisation, where each line in the event chart constitutes one hit of the query for a single patient. There is thus one row for each hit in the database, and these are sorted by patient. The background of the visualisation contains grey-scale rectangles representing non-matching events, where the amount of greyness reflects the number of events at a given point. A tabular view of the results is also available.

³In fact so recent that it has not yet been published; we came across this project, very similar to ours, after having developed and tested our solution, and after most of this thesis was written.

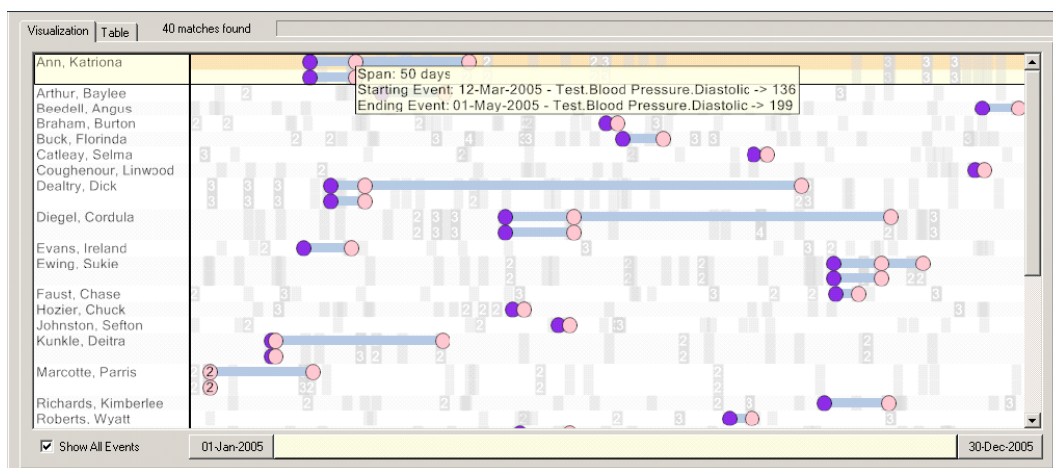


Figure 2.11: Query result visualisation, from [FKSSo6]. Hits of the queries are shown as horizontal bars, vertically collected by patient. The background indicates context by showing events: The darker the marker, the more events happen at that time.

2.4.5 Comparison

The visualisation used by Fails et al can remind of an event chart showing multiple lines per history, one for each hit of a temporal query. However, the visualisation shows only the time spanned by the search hits, as opposed to the traditional event chart showing the entire histories. Also, events not part of a search hits are only counted in the design of Fails et al, while an event chart typically treats all events selected for display equally.

Concerning the relation between event charts and life course visualisations, both show history information in a timeline view. While event charts (and the design of

Fails et al) show many histories in the same view, the life course visualisations show much more information for the one history they show.

2.5 Evaluation methods

When proposing a new design, it needs to be evaluated with respect to its fitness for the purpose for which it was proposed. Also, it is useful to relate the design to relevant taxonomies. In such an evaluation, it is important, as pointed out by Freitas et al, to be aware that an evaluation of an interactive information visualisation application will end up testing three different aspects [FLC⁺02]:

- *Visual representation usability*: The fitness of the visual representation for a given purpose.
- *Interface usability*: How well the user interface supports the objective of the visualisation.
- *Data usability*: To what degree the data supports the tasks to be tested.

This section describes the following topics:

1. *Empirical evaluation*: Methods for performing controlled experiments for evaluating a visualisation or user interface.
2. *Expert reviews*: Evaluation by comparison to guidelines and recommended practice.
3. *Relevant taxonomies*: Description of two taxonomies that will be used to position the proposed design.

2.5.1 Empirical evaluation

In scientific studies, empirical evaluation plays an important role. It is possible to perform such an evaluation of the usability of a computer system by measuring performance on a set of tasks with or without the new system. However, this means that there has to be a set of tasks to be performed. In the case of an explorative application, like ours, the goals for interaction are difficult to measure. This section introduces two relevant techniques: Usability tests, which are in common use, and insight-based evaluation, a novel method for quantifying the insight gained into a data set using an explorative tool.

USABILITY TESTS

Usability testing involves assigning tasks to a group of users and observing them while solving these tasks using the interface to be evaluated [Shn98]. Typically the users are directed to “think aloud”, explaining what they do, and why. The sessions are often videotaped, and times and error rates for solving the tasks can be measures.

A similar approach could be used in the evaluation of visualisations, and Martins et al report one such study evaluating a “Tool for Intelligent Query and Exploration of Patient Data” [MS⁺04]. In this study, eight test subjects were asked to answer ten questions of increasing difficulty related to a patient history. The questions were answered using a paper-based presentation, a spread-sheet program, and the new tool to be evaluated. Answering times and error rates were measured, and the differences tested for statistical significance.

INSIGHT-BASED EVALUATION

Saraiya et al [SND05] describe a method for evaluation of information visualisations that strives to quantify insight into the data according to several criteria representing characteristics of insight. They define an *insight* to be a single observation about the data made by a user. For each insight reported, several characteristics are quantified and recorded by domain and visualisation experts. These characteristics include measures such as time to acquire the insight, domain value (on a scale of 1-5), correctness, and detailedness of the observation. In this framework, the insights that generate new hypotheses are regarded as the most valuable.

If such an evaluation is going to be carried out, the criteria for quantifying insight must be adapted to the domain in question; Saraiya et al performed their research within the field of biology, we would need to apply this method to the general practice domain. One step in the process of adapting the criteria is to perform a pilot test to discover relevant kinds of insights. This thesis performs a small case study, which may serve as a pre-pilot that equips us with the necessary knowledge to perform the pilot.

2.5.2 Expert reviews

Another way to perform the evaluation, is to assess the product with respect to guidelines or heuristics. This is an evaluation method often used for evaluating user interfaces [Shn98], where a usability expert reviews the screens of a program to assess their usability. Such experts ought to be external to the development team, but in this thesis we will perform the evaluation ourselves, relating our prototype to a set of criteria. This will highlight missing functionality compared to other information visualisation designs. At a later stage, a visualisation expert should be called for.

Freitas et al have developed heuristics for interactive information visualisations, encompassing both the visual representation and interactivity aspects [FLC⁺02]. These criteria are aggregated into larger groups (see figure 2.12):

Limitations

Geometric and visual constraints, such as the available display area or the maximum number of visual items that can fit into the visualisation.

Cognitive complexity

The number of dimensions that are shown simultaneously, and the density of the information displayed (e.g. number of items). Also, these criteria include measuring the relevance of the data that is shown.

Spatial organisation

How information is laid out on the display is important, and the criteria related to spatial organisation address how easy it is to find information, and how the overall distribution of the elements is presented and related to the surrounding context. As presented by [FLC⁺02], this point seems to address what focus+context mechanisms (“Zoom” in section 2.3) try to remedy.

Information coding

How the information to be displayed is mapped to visual representations. Also, the use of realistic techniques such as depth cues or photo-realistic rendering is assessed.

State transition

When the user interacts with the application, this may lead to a change in the visualisation. How long this change takes to perform, and how much the visual organisation changes both influence the user’s sense of orien-

tation in the data set and perception of what the change did to the visual representation. It should be observed that even substantial changes can be difficult to detect, because of change blindness (section 2.3.2).

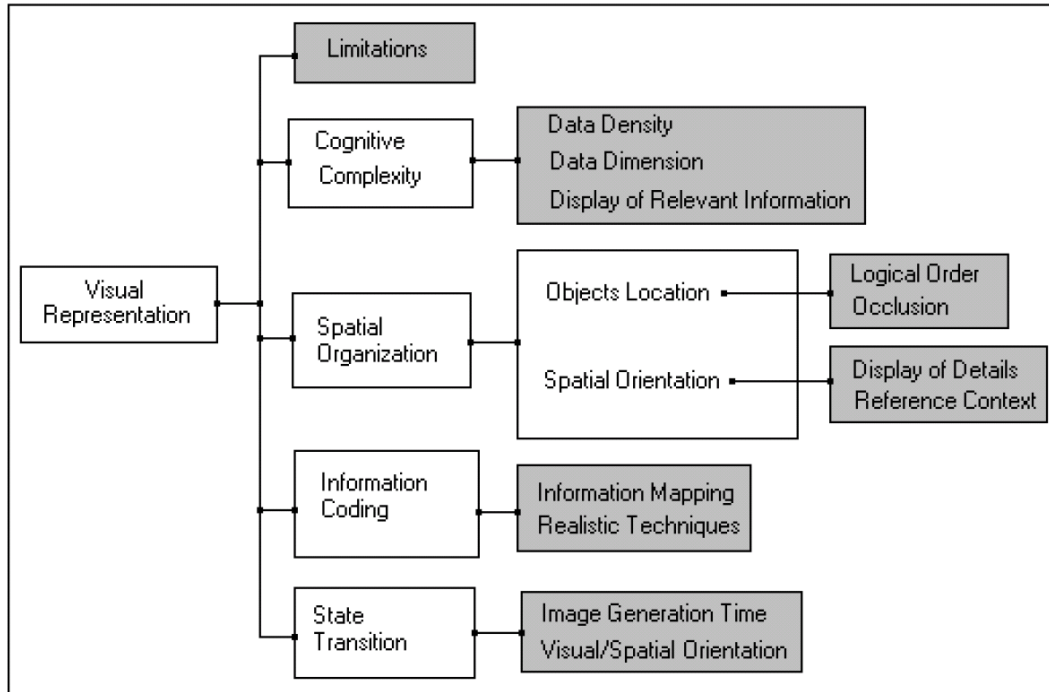


Figure 2.12: Heuristic criteria for evaluation of information visualisations, overview. Figure taken from [FLC⁺02].

Freitas et al also list criteria for evaluating interaction mechanisms, as shown in figure 2.13:

Orientation and help

The user should be given the control over the level of detail in a display, and the system ought to be forgiving by providing undo functionality. Furthermore, these criteria include control over the display of additional information; Freitas et al exemplify this by: “the path a user followed while navigating in a complex structure” [FLC⁺02].

Navigation and querying

Several criteria are given that relate to navigation and querying functionality, such as selection, manipulation of the point-of-view and the geometry displayed, searching and querying, and zoom (“Growing” in figure 2.13).

Data set reduction

Three evaluation criteria are given concern how the data set can be reduced: Filtering (temporarily hiding information), clustering (aggregating visual representations) and pruning (cutting off information).

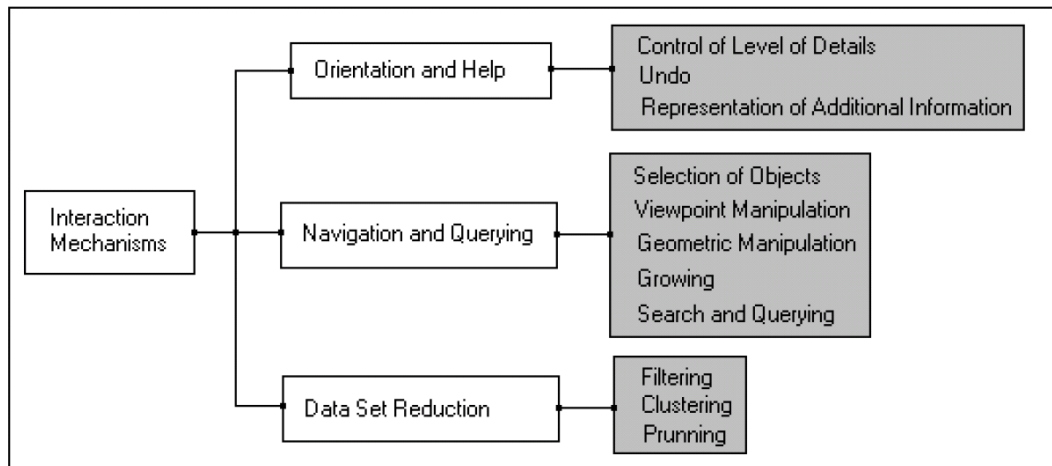


Figure 2.13: Heuristic criteria for evaluation of interaction mechanisms, overview. Figure taken from [FLC⁺02].

2.5.3 Relevant taxonomies

Taxonomies are used for classification, and they may help see the relation between different techniques as well as serving as useful tools in positioning a new design among the existing. In addition, having the taxonomies in mind while developing a new design can highlight missing functionality that should be included.

SHNEIDERMAN’S TASK BY DATA TYPE TAXONOMY

An often cited taxonomy is provided by Ben Shneiderman in his article: *The eyes have it: A Task by Data Type Taxonomy for Information Visualizations* [Shn96]. This taxonomy describes seven data types:

- *Linear data:* Lists, sequences or text.
- *2- and 3-dimensional data:* Spatial data, such as maps.
- *Temporal data:* Histories, where data items have start- and end times. The entries can be overlapping.

- *Multidimensional data*: Data where each data point have many values, such as in a tuple from a relational database.
- *Tree data*: Hierarchal data, such as organisation charts.
- *Network data*: Data that are arbitrarily interrelated, such as computer network interconnection diagrams.

In addition, seven basic tasks that ought to be possible are described (in verbatim from [Shn96]; see the detailed description in section 2.3.3):

- *Overview*: Gain an overview of the entire collection.
- *Zoom*: Zoom in on items of interest.
- *Filter*: Filter out uninteresting items.
- *Details-on-demand*: Select an item or group and get details when needed.
- *Relate*: View relationships among items.
- *History*: Keep a history of actions to support undo, replay, and progressive refinement.
- *Extract*: Allow extraction of sub-collections and of the query parameters.

More about the seven tasks can be found in section 2.3. According to Freitas et al, this taxonomy has also been used in expert reviews of information visualisations [FLC⁺02].

A TAXONOMY FOR VISUALISATIONS OF TIME DEPENDENT DATA

Müller and Schumann describe another taxonomy in their survey article from 2003 [MS03], where they classify data points as belonging to one of the following categories (or as tuples of different types):

- *Nominal*: Data items belong to one of a set of named categories.
- *Ordinal*: Data items can be ordered, i.e. there is some kind of ordering that applies to them.
- *Quantitative*: Data items have a numerical value.

In addition, the time axis itself is subject to characterisation⁴, as (italicised text copied in verbatim):

- *Discrete time points vs. interval time*: Discrete time points have no duration, while intervals are defined from a start point to an end point.

⁴The taxonomy is due to a study by Frank from 1998.

- *Linear time vs. cyclic time*: Whether time is running from past to future or is cyclical (as in a time axis showing the four seasons).
- *Ordinal time vs. continuous time*: If the time axis is scaled ordinal, only relative ordering of events is possible. With a continuous time axis, the time difference between events is possible to calculate.
- *Ordered time vs. branching time vs. time with multiple perspectives*: Ordered time describes a single course of events, while branching time is capable of describing multiple alternative scenarios. Time with multiple perspectives allows multiple data points for each time step, enabling description of parallel processes.

2.5.4 Comparison

The evaluation methods introduced in this section represent three different approaches with individual strengths and weaknesses:

1. *Empirical evaluation*: Through empirical evaluation, hypotheses can be confirmed or rejected, and this is thus the most solid approach to evaluation. However, there are two serious drawbacks in our setting: The cost (both in time and resources) of planning and carrying them out, and the difficulty of evaluating an explorative design quantitatively. Saraiya et al remedy this last problem through their framework for insight-based evaluation, but this would in our case require at least two studies: One to adapt the framework to our settings, and another to do the actual evaluation.
2. *Expert reviews*: Comparing a design to guidelines can be useful in order to detect weaknesses. Ideally, several external experts should be called upon, to diminish the impact of biases. These evaluations are considerably cheaper and easier to perform than empirical, but they have less precision than usability tests [GHS99].
3. *Taxonomies*: Although not an explicit form of evaluation, relating a design to taxonomies can be useful to position a design among the existing, and the process of assessing the design in relation to a taxonomy may reveal weak spots in the same way as in an expert review.

Gabbard et al suggests a process of iterative evaluation and improvement, where expert reviews are performed first because of their low cost, followed by formative user-centred evaluation. The formative evaluations are carried out by having users use the program “thinking aloud”, while being observed by the developers. This is very simi-

lar to our case study. After several iterations of expert reviews and formative evaluations, a final empirical evaluation is performed, comparing the new system to mature alternative tools to measure the improvement [GHS99]. The work performed in this thesis can be seen as one and a half iterations of the above: Development of a tool, with guidelines in mind, followed by a case study with a user, and then a discussion of the results from the case study and of the design with respect to different sets of guidelines. No empirical evaluation is performed at this stage.



Proposed design

(chapter three)

With basis in and references to the work described in the preceding chapter, this chapter proposes a visualisation with accompanying interactive features. The visualisation shows each patient history as a bar annotated with symbols representing the events in the history, and interval concepts shown by background colourings. Interactive operations on this diagram include extraction of sub-collections, sorting and aligning histories, filtering events, and searching for temporal patterns. The proposed visualisation is implemented in a Java prototype; a screenshot of this prototype is shown in figure 3.3 (page 55), along with a description of its user interface. Appendix E.1 explains how to run the program, which is provided as a digital appendix to this thesis.

This chapter is structured as follows:

1. *Data model*: An Entity-Relationship (ER) model for the data that is to be visualised, along with descriptions of how this is extracted from the patient database described in section 1.5.
2. *A static visualisation of patient histories*: Explanation of the visualisation implemented in the prototype in terms of its constituent visual elements.
3. *Adding interaction*: Exploration-supporting interactive features of the prototype.
4. *Temporal query language*: The query language used to perform some of the interactive operations, and a set of algorithms for its operators.

3.1 Data model

To speed up drawing and become more independent of the database schema of the EHR tool from which the data is collected, all content to be visualised or queried is pre-loaded into a data structure of Java objects. This structure is modelled in figure 3.1, as an Entity-Relationship (ER) diagram describing which entities are represented, and how they relate. Note that the diagram only shows the entities that are visualised. See appendix C, figure C.1, for a complete diagram of all entities handled by the system.

In figure 3.1, a `HistoryCollection` represents the entire content of the visualisation, consisting of a collection of `History` instances. Histories are defined by their entries happening at given points in time, represented by `Entry` in the diagram.

The entries themselves are either intervals, defined by their start and end times, or events that happen at a given time and have no duration. Medications are the only intervals that are considered and visualised for the time being, but this could be extended to include more interval information from the database, such as sickleaves, or medical abstractions. Concerning the latter, this could be notions such as “high blood pressure”, “HIV positive”, or it could be intervals deduced from the other data; a report by Nordbø shows how sequences of diagnosis codes can be abstracted into intervals and visualised [Nor05]. Concerning point events, these are either contact diagnoses or tests. Even though many tests are read from the database, currently only blood pressure measurements are shown in the main visualisation – the other results are available by “details-on-demand”, in text form.

When it comes to the representation of time, it is worth pointing out that all timestamps from the database are truncated to the precision of days, and all events occurring on the same day are assumed to occur simultaneously. This is because one day is the smallest granularity at which the database is accurate: Many of the entries are dated correctly, but the time of day is midnight – which is unlikely to be correct. Entries with a clearly invalid date (prior to the birth of the patient) are ignored.

3.1.1 Conceptual hierarchies and regular expressions

There are two types of data that are categorised according to hierarchies in our data model: Diagnoses and medications. In the prototype, we select points in these hierar-

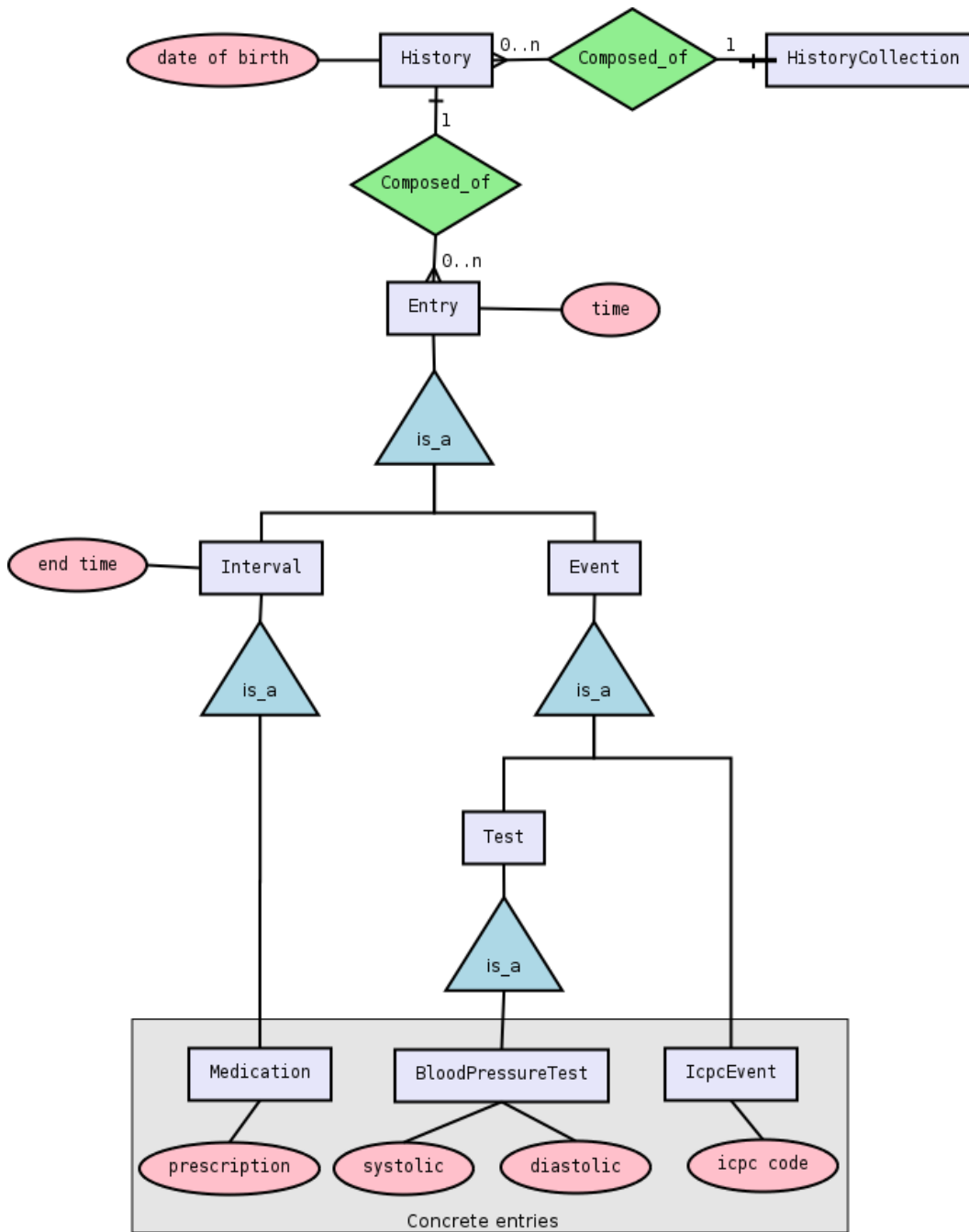


Figure 3.1: An ER model for the data represented and used by the visualisation prototype. The blue triangles indicate inheritance, and only the leaf nodes of the “Entry” inheritance tree are instantiated in the visualisation. This diagram represents a simplification, and it does not show all detailed attributes (e.g. “prescription” contains many sub-fields) or entities that are not being visualised. See figure C.1 for a more complete diagram, also showing entities that are not part of the visualisation.

chies using regular expressions, but in a later version we should consider modelling the hierarchies.

DIAGNOSES

The diagnoses in the database are coded in the International Classification of Primary Care (ICPC) [WON98], a system where all codes are given with a chapter (single letter) and a two-digit code. In addition, another dimension is implied by the categorisation of codes within each chapter into the categories:

- Process codes
- Symptoms/complaints
- Infections
- Neoplasms
- Injuries
- Congenital anomalies
- Other diagnoses

Another categorisation is given by Grimsmo [Gri], where the classification is oriented by problem rather than organ system. This categorisation is not directly supported in the prototype, but the groups can be specified with regular expressions. In a later version, we should consider supporting specification of these groups directly.

Parts of the ICPC hierarchy are given below:

```
...
A General and unspecified
...
  A07 - Coma
  A08 - Swelling
  A09 - Sweating problem
...
B Blood, Blood Forming Organs and Immune Mechanism
...
  B73 - Leukaemia
  B74 - Malignant neoplasm blood other
...
K Cardiovascular
...
  K85 - Elevated blood pressure
```

```

K86 - Hypertension uncomplicated
K87 - Hypertension complicated
...
Z Social Problems
  Z05 - Work problem
  Z06 - Unemployment problem
  Z07 - Education problem
...

```

MEDICATIONS

Medications are coded in the Anatomical Therapeutic Chemical Classification System (ATC) [WHO05]. This is a five-level hierarchy with the following levels:

1. *Anatomical group*: One letter.
2. *Therapeutic main group*: Two digits.
3. *Therapeutic/pharmacological subgroup*: One letter.
4. *Therapeutic/pharmacological/chemical subgroup*: One letter.
5. *Chemical substance subgroup*: Two digits.

Parts of the ATC hierarchy are given below:

```

...
C07      BETA BLOCKING AGENTS
C07A     BETA BLOCKING AGENTS
C07A A   Beta blocking agents, non-selective
          C07AA01   Alprenolol
          C07AA02   Oxprenolol
          ...
C07A B   Beta blocking agents, selective
          C07AB01   Practolol
...
C08G     CALCIUM CHANNEL BLOCKERS AND DIURETICS
C08G A   Calcium channel blockers and diuretics
          C08GA01   Nifedipine and diuretics
C09      AGENTS ACTING ON THE RENIN-ANGIOTENSIN SYSTEM

```

```
C09A      ACE INHIBITORS, PLAIN
C09A A    ACE inhibitors, plain
...
```

REGULAR EXPRESSIONS

We use regular expressions to describe subsets of the above hierarchies. The main motivation for this is simplicity: Regular expressions are readily supported by our programming environment (Java), and with a regular expression one may easily refer to any branch of the hierarchies by listing the first few letters or digits and appending a wildcard. Such specifications may be combined using the disjunctive construct of regular expressions; so to specify diagnoses concerning the eye (F) or ear (H) one may specify the regular expression: `F.*|H.*`. While being a useful tool for computer scientists, general practitioners cannot be expected to be acquainted with regular expressions. This means that in a future version, a graphical user interface should be built.

Regular expressions are also used for extraction of some of the available data, as explained below. However, this extraction is limited because of differing conventions and many typing errors in the free-text.

3.1.2 Adoption of the available data

The database consists of more than 3,000 fields, many of which are free-text. This means that, in addition to choosing a meaningful selection of fields, parsing of free-text is necessary. Also, there are several problems with the data (see section 1.5.2) that need to be remedied prior to visualisation.

EXTRACTION OF BLOOD PRESSURE MEASUREMENTS

Blood pressure measurements are extracted using the regular expression:

```
[Bb][Tt][^0-9]*[0-9]+/[0-9]+
```


This expression matches the string “BT” (case-insensitively) followed by any sequence of zero or more non-digits and a construct on the form “<integer>/<integer>”.

FILLING IN MISSING ATC CODES

Some medications miss an ATC encoding. Most of these are repaired by looking up ATC codes from other prescriptions with the same the medication name. In a few cases, several codes are returned. This is resolved by choosing the ATC code most frequently used to describe the drug. However, not all ATC codes can be filled in this way, meaning that there will be medications that the prototype miss in the filtering and colouring process.

After performing the procedure outlined above, 13,881 out of 241,307 prescriptions (5.75%) lacked an ATC code. This is not an exact number, because many “prescriptions” in the database are actually items like bandages and syringes, which do not have an ATC code. Filtering out the most common non-drugs brings the number of prescriptions lacking ATC code down to 9,445 (3.91%).

DEDUCTION OF MISSING MEDICATION CESSATION DATES

Medications represent an interval concept: At a given day, the doctor prescribes a medication for a patient. This medication is limited in quantity, and its daily consumption is generally predictable (e.g. “take two pills before dinner”). Thus, it should be possible to deduct for how long the prescribed medication would last. The box below outlines our procedure for deducing cessation dates for medications.

Before any parsing is performed, the following rules are applied:

- If the medication has a given cessation date, this is used.
- Prescriptions meant to last a “long” time (they are stored in a separate table in the database) are defined to last for one year.
- Medications that are taken by need are not processed, since they do not have a predictable rate of consumption.
- Medications like ointments are not treated, because it is not possible to guess how long for example 50 grams of ointment will last when it is applied to the nose twice a day.

If none of the above rules apply, the following sequence is performed:

- The quantity field is a free-text field supposed to hold a single number. This is parsed. If the quantity is unparseable, the deduction is given up.
- Then, the daily consumption is attempted guessed from two free-text sources:
 - Usage instructions: The usage instructions listed on the packet handed out to the patient is parsed and checked against a database of examples of usage instructions.
This database is constructed by transforming all usage instructions in the database to a “standard form”: For example, common abbreviations for pill (“tablett”, “tab”, “tbl”, etc.) are transformed into one particular form. The standard strings that are equal are collected, and these strings are manually read and marked with the number of doses consumed each day. Last, the usage instructions in the database are then updated with the consumptions from their corresponding standard forms.
 - Short instructions: There is also a field for listing usage on a short form, such as “1+1+1”. If the standard usage instruction is unparseable, this field is tried for some common short forms.

If the above fails, the medication is marked as un-parseable.

The above deduction scheme for prescription cessation dates has limited generalisability because the example database used is fine-tuned for our data. Whether our transformation to a “standard form” is valid for other databases, is not known, and would depend on the individual doctor’s documentation habits. Also, this process is time-consuming – especially because of the many typing errors, making the transformation to the “standard form” less efficient by creating many more unique instances. We consider the time spent to be acceptable for a proof-of-concept prototype like ours. This would not be an issue if the database was more structured (field-oriented), storing dosages and daily consumptions in a consistent manner. Hopefully, future health record databases will be less free-text oriented than the present.

3.2 A static visualisation of patient histories

Given the available data and the data model, a suitable graphical representation was to be found. Our requirements were as follows:

- *It shall be possible to compare a group of patients:* This is directly derived from

the main goal of this thesis.

- *The dimension of time shall be preserved and shown along an axis:* Our experience from the previous project was that the notion of time is very important, and the lacking support for this was a weak spot for our previous prototypes. Hence, this was a key feature for our new design.
- *Both point events and intervals shall be represented:* Previous prototypes had been concerned with visualisation of point events. Since the EHR contains interval concepts as well, it was natural to support this in order to include more of the data that was available. This relates to the future work listed in[Nor05].

To satisfy these requirements, we decided to depict the histories as timelines annotated with icons for point events and background colourings for intervals. Different classes of events are plotted in separate vertical subdivisions of the bar. Time is spaced evenly along the bar, and all bars share the same scaling constant.

An axis is also provided in the diagram. This axis initially shows calendar time, but when the bars are aligned on a common pattern (see section 3.3.3), the display changes to reflect time relative to the point in time on which the histories are aligned.

Figure 3.2 shows the visual representations to be explained in the subsequent sections.

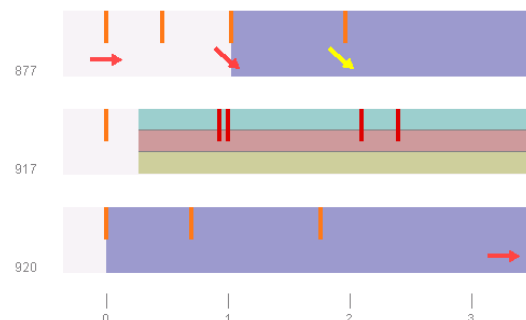


Figure 3.2: The visual representations explained: Vertical rectangles represent contact diagnoses, arrows represent blood pressure measurements, and the background colours signify the presence of medications. When several medications are in use at once, the space is divided evenly between the medications (shown in the middle bar, patient 917).

3.2.1 Events

Events appear at a given date and are assumed to have no or very short duration (compared to the length of the history). The representation should thus be that of an icon or glyph.

DIAGNOSES

During the development of the prototype, the question of how to represent contacts with several diagnoses was raised. Simply drawing an iconic representation would lead to overdrawing and only the last diagnosis drawn being visible. We considered several alternative solutions to this problem, including jittering (moving each icon a small, random amount) and subdividing the representation. In the end, the latter alternative was chosen, and the final representation was decided to be a coloured rectangle. This rectangle is subdivided vertically into smaller rectangles coloured according to the diagnoses occurring at the date in question. Colours are assigned based on regular expressions on the ICD code.

Diagnoses are presented as follows:



Here, the colour blue signifies stroke, while orange shows hypertension. Some of the rectangles are subdivided to show that there are several diagnoses at that day – the details are available in a separate display area when the mouse is placed over the representation.

TESTS

Assigning iconic representations to tests conveys the information that the test was performed, or that its result was received. In an early stage of development, the prototype used a small glyph to represent each test.

3.2. A static visualisation of patient histories

While this may be sufficient in some cases, it is beneficial if as much information as possible can be read from the diagram at a glance (as long cognitive limits are observed). Incorporating more information means mapping test results, and possibly trends, to features of the representation¹. In light of this, the icon for blood pressure tests was changed to an arrow whose direction shows the trend (up, down or unchanged since the last measurement), while its colour reflects a classification of the value (normal, mild hypertension or severe hypertension). The angular difference between any two trend indications is at least 45 degrees, which is above the cognitive limit for absolute angle detection at 30 degrees.

¹Section 5.3.1 discusses this in more detail.

Blood pressure measurements are represented as follows:



Here, red arrows signify moderate to severe hypertension, while yellow arrows show mild hypertension. The green arrows – normal blood pressure – at the end of the bar indicate that the treatment of this patient's hypertension was successful. These colourings are based on Norwegian limits for classification of hypertension [ASo6]. The direction of the arrow indicates the trend in diastolic blood pressure.

3.2.2 Intervals

In the prototype, medications are the only intervals that are visualised. They are shown using colouring of the background. Medications can be assigned a colour or texture based on a regular expression on the ATC found in their prescriptions.

Medications are represented as follows:



Here, a patient initially has no medications. Then, three medications are successively prescribed. The most recent medication is drawn at the top.

3.2.3 Axes

To make it possible to address individual patients, patient ID numbers (taken from the database) are shown along the vertical axis. The horizontal axis has two modes:

1. When the diagram is not aligned, the axis shows calendar time (the actual dates).
2. In an aligned diagram, the axis shows the number of months before and after the alignment point.

Showing calendar time:



Showing relative time:

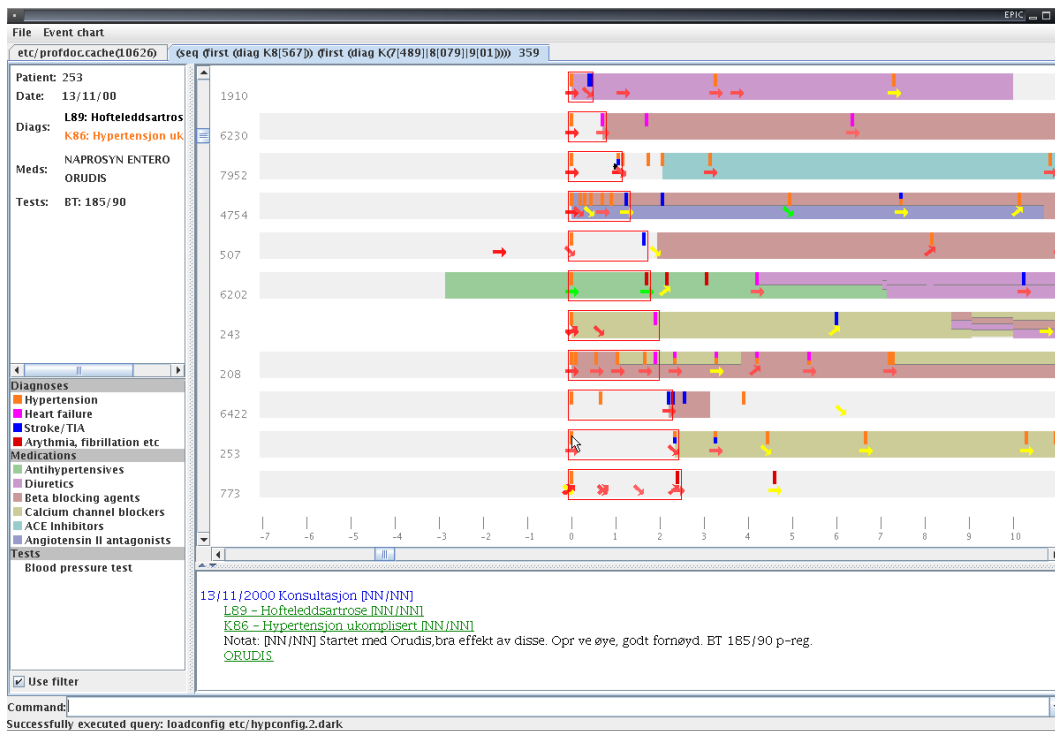


3.3 Adding interaction

Computer visualisations become exploration-supportive from being interactive, rather than just providing a static diagram; the diagram itself acts as a dynamic interface to the data, and its parameters can be changed on the fly in response to the user's ideas and interests.

Figure 3.3 shows the main window prototype along with an explanation of its user interface. In the following sections, different interaction mechanisms are explained.

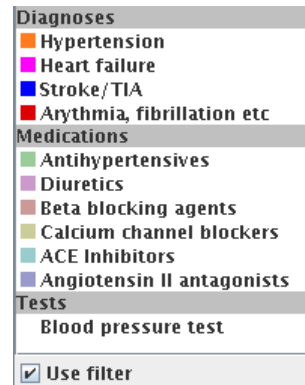
3.3. Adding interaction



(a) The prototype main window



(b) The information pane



(c) The colour legend

Figure 3.3: The prototype user interface. (a) shows the prototype running a visualisation of hypertension patients with complications, sorted on the time from hypertension to complication. (b) shows a magnification of the information pane that reflects the content under the mouse cursor, while (c) explains the colours used in the diagram and provides a quick way of toggling filtering on the event level. The text area in the lower part of the window displays the contact note (if any) for the contact under the mouse cursor.

3.3.1 Selecting information to visualise

Crowding causes a diagram to become difficult to interpret and can make patterns and abnormalities harder to spot; see figure 3.4(a) for an example of how some histories look without filtering. To remedy this, filtering is employed to remove or hide certain parts of the visualisation. In our visualisation, filtering can be employed on the level of histories or on that of events.

On the event level, three sorts of filtering are possible:

- *Contact diagnoses*: The ICPC coding system implies a hierarchy, and filtering can be applied at any point in this hierarchy using regular expressions. Figure 3.4(b) shows the histories in figure 3.4(a) with all diagnoses and medications not related to hypertension removed.
- *Prescriptions/medications*: Medications can be classified with respect to the ATC coding hierarchy. Regular expressions on this code are used in the prototype.
- *Tests*: Tests can be filtered according to test type, possibly with constraints on which values are considered interesting. The current prototype only allows filtering on test type.

On the level of histories, the prototype allows inclusion or exclusion of histories based on expressions in a temporal query language. See sections 3.3.3 and 3.4 for details. Figure 3.4(c) modifies figure 3.4(b) by removing histories not containing a hypertension diagnosis, with the result of bringing more of the relevant patients into view.

3.3.2 Information availability: Details-on-demand

²This represents the final step in Shneiderman's information-seeking process, see sections 2.3.3 and 2.5.3.

Having access to detailed information² is crucial when inspecting a visualisation that represents a simplification of the data. In the case of the Electronic Health Record (EHR), this detailed information is found in the contact notes and in the fields for contact diagnoses, prescriptions and tests.

Since the user, a doctor, is used to read a textual record, the prototype provides an interactive relation from the diagram to the record. This is realised by having the user click in the diagram to open a record note view scrolled to the entry nearest the point that was clicked (via a pop-up menu). Figure 3.5 shows the record viewer, modelled after the record view in *ProfDoc*, an EHR system in widespread use in Norway. The imitation of *ProfDoc* is believed to make the record easier to read for the test subject

3.3. Adding interaction

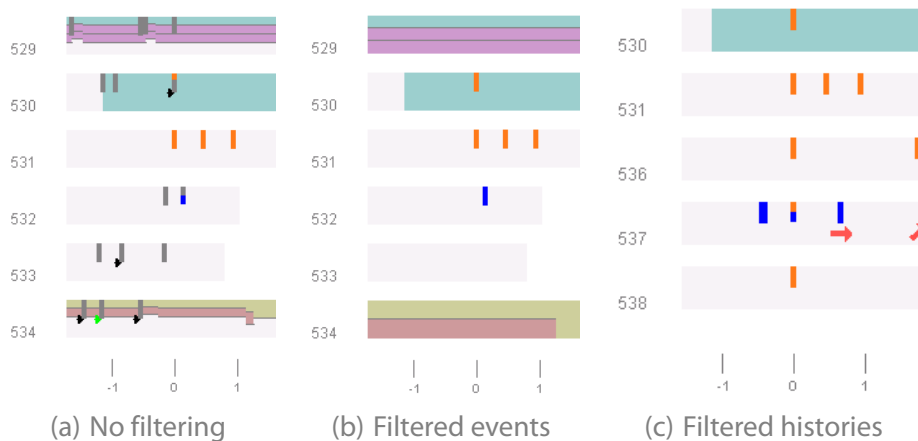


Figure 3.4: Filtering illustrated: (a) shows a small part of the entire database, with all events shown. Diagnoses and medications related to hypertension, and blood pressure tests, are coloured. (b) shows the result of hiding all diagnoses and medications not related to hypertension. (c) shows the result of extracting hypertension patients. Note that now five patients with hypertension are displayed, compared to two in (a) and (b).

in our case study, who is familiar with the tool.

Furthermore, since the chosen iconic representations are not self-explanatory, and their representations may overlap, a brushing technique is employed: A panel is dynamically updated to show the information represented at the point of the mouse cursor. This includes summary information such as diagnosis codes and medication names, as well as the full contact note. In figure 3.3, the lower panel shows the full contact note for the contact that is pointed to with the mouse, while the upper left area show a summary of all diagnosis codes, medications and tests assigned at that contact.

3.3.3 Query-based operations

Making the visualisation dynamic and explorable involves defining how the presentation can be changed by the user. For event charts, the extensions proposed by Lee et al [LHDoo] include aligning the histories on a common event, and sorting or grouping the histories. This section describes and exemplifies these and other operations that are specified with a temporal query language, to be described in detail in section 3.4.



Figure 3.5: The record viewer showing parts of a health record. All contact notes have been removed as part of anonymisation – when using unanonymised data they would appear after the text “Notat: [NN/NN]”, with the name of the responsible physician instead of “NN”.

ALIGNMENT

Alignment is important in order to bring interesting patterns close to each other for inspection of similarities and differences. The align operation supports specification of complex temporal patterns, and the prototype automatically highlights the interval matched by the alignment criterion. Besides being a tool in its own right, the alignment operator is invoked by the other operations to align newly extracted sub-collections, or to line up the sort criterion highlighting the difference between areas of interest. Figure 3.6 illustrates the effect of alignment.

SORTING

Sorting adds more information to the diagram by ordering the histories according to a user-specified sort criterion. This order can be based on attributes of the patient, such as age, or on properties of the histories, such as time from a diagnosis to the prescription of a certain medication. A use for the latter kind of criterion, i.e. sorting on the distance between two query hits, is to gather similar cases when the length of the

3.3. Adding interaction

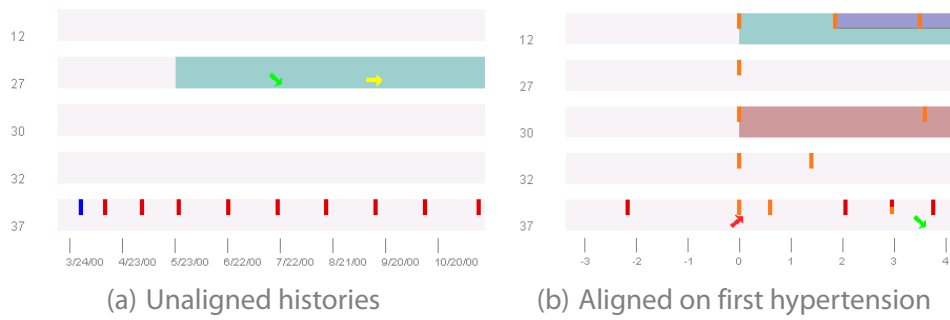


Figure 3.6: Alignment illustrated: (a) shows five patients suffering from hypertension, using calendar time. (b) shows the same five patients, now aligned so that the first occurrence of a hypertension diagnosis is lined up, and the axis now shows number of months before and after the diagnosis.

region of interest is important. This is illustrated in figure 3.7, showing patients suffering from complications of hypertension sorted on the time between hypertension and complication.

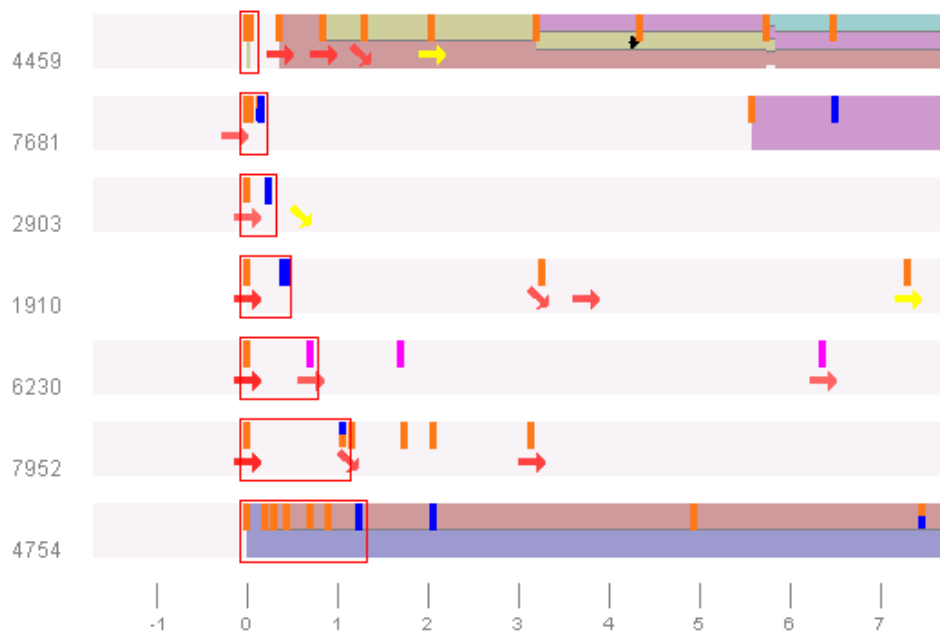


Figure 3.7: Result of applying the sort operation to a group of patients having diagnoses known as complications to hypertension. The patients are sorted on the time from the first hypertension diagnosis to the first complication. A red rectangle indicates this period.

SELECTION

Selection enables the user to isolate the histories of interest by extracting a sub-collection, or by removing all histories matching a given query. This reduces the amount of information and is absolutely necessary to get down to a manageable number of histories to view: In the available database, there are 10,515 histories, and 778 out of these contain a hypertension diagnosis. Even the latter represents a considerable number of histories. In our case study (chapter 4), the analysed collections contained from 5 to 35 histories. Selection is illustrated in figure 3.4, as the difference between figures 3.4(b) and 3.4(c).

SEARCH

Exploring the diagram also involves looking for patterns, or investigating the frequency and position of an interesting pattern. For this purpose, there is a search tool that locates all instances of a given pattern, highlights these with a red frame, and lets the user examine them systematically. Such red frames are shown in figure 3.7, and the prototype enables cycling through the matches using keyboard shortcuts. The available navigation is: Next/previous hit in the current history (jumps to next/previous history if no such hit), and next/previous history (wraps from bottom to top if “next history” is invoked with the last history, and vice versa).

3.4 Temporal query language

As described above, the prototype employs a temporal query language for specification of several of its exploration-supporting operations. This language enables the user to specify patterns based on events and intervals in the histories, and on the temporal relations between them.

Figure 3.8 shows how a simple query is tokenized and parsed: The tokenizer splits the query into primitive symbols that are interpreted by the parser to build a query tree. Running a query amounts to extracting matches from the root node of this tree. The root node, and the internal nodes below it in the tree, will query its children and combine the information to produce a match. On the leaf node level, the actual searching in the histories is performed.

We desired lazy execution of the queries, and as a consequence, the queries are actually run in a structure of iterators mirroring the query tree (see the section on algorithms, section 3.4.2). Figure 3.9 shows an illustration of how the query constructed in figure 3.8 is run.

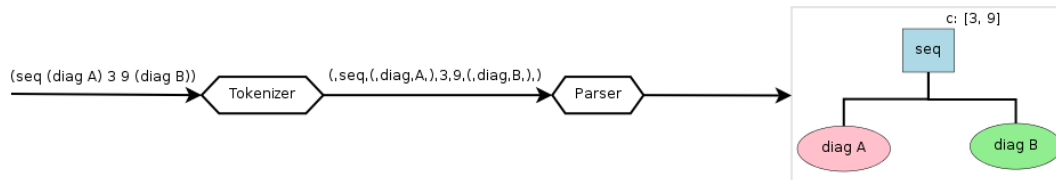


Figure 3.8: Tokenizing and parsing a query: The tokenizer transforms a string representing the query into a list of symbols that is interpreted by the parser to create an object structure. See figure 3.9 for an illustration of how this query is run on a history.

3.4.1 Language definition

The query language is recursively defined: At the bottom level, primitive elements are capable of locating single events or intervals in the histories. These elements can act as queries in their own right or be combined using compound elements that combine or filter the result of queries to express temporal relationships. This section describes the query language, both in form of a grammar on Backus-Naur Form (BNF), and in natural language. Each operator is illustrated with an example, and the last subsection contains an example of a more complex query. More examples are to be found in the presentation of the case study, chapter 4.

PRIMITIVE ELEMENTS

In the leaf nodes of a query tree, the primitive elements of the query language match individual events.

The grammar for the primitive events is as follows:

```
<EType> ::= (icpc <RegEx>
           | (atc <RegEx>
           | (test <TestType>)
```

```
| (birth)
| (now)
| (beginning)
| (end)
```

Here, <RegEx> refers to a regular expression, and <TestType> is a string denoting the name of a test (e.g. bp for blood pressure measurements).

The following primitive elements are defined:

icpc Finds all diagnoses matching a regular expression on ICPC code (described in section 3.1.1). For example, the query (*icpc* K.*) finds all codes related to the cardiovascular system.

atc Finds all prescriptions matching a regular expression on ATC code (described in section 3.1.1). For example, the query (*atc* C07.*) finds all beta blockers (group 07 of the cardiovascular drugs (C)).

test Finds all tests of the given type. Currently only blood pressure measurements are recognised; the query (*test* bp) finds all blood pressure measurements.

Dates Static information about the history or the environment are provided for the following:

- The patient's date of birth: (*birth*).
- Current date: (*now*). This is implemented by returning the latest date in the entire collection of histories, since the database we use is no longer being updated.
- Beginning of a history (registration with the physician's office): (*beginning*).
- End of a history (last recorded event in the history): (*end*).

COMPOUND ELEMENTS

Compound elements combine the results from primitive elements and other compound elements to specify temporal properties of the pattern to be found.

The grammar for compound operators is as follows:

```

<Pattern> ::= <Etype>
           | (seq      <Pattern> [<Int> <Int>] <Pattern>)
           | (and      <Pattern> <Pattern>)
           | (or       <Pattern> <Pattern>)
           | (windowwith <Pattern> <Int>)
           | (windowwithout <Pattern> <Int>)
           | (first    <Pattern>)
           | (merge    <Pattern> [<Int>])
           | (startof  <Pattern>)
           | (endof    <Pattern>)

```

Here, <Int> denotes an integer.

There are three elements that can be described as compositional, used to combine two queries:

- seq* Find a non-overlapping sequence of matches from two given queries, possibly with constraints on their separation. The constraints must be zero or positive, and it is required that the second constraint is equal to or larger than the first. An example: `(seq (icpc K90) 0 3 (icpc A96))` will match a patient that dies (code A96) within three days after having a stroke (K90). This operator can also be used to specify simultaneousness (two events occurring on the same day) by setting both constraints to zero.
- and* Identify regions where parallel conditions hold, i.e. return the cartesian product of the hits generated by two queries. Example usage: `(and (icpc K87) (icpc F83))` matches patients who have hypertension with organ complications (K87) and retinopathy (F83; can be a complication of hypertension).
- or* Identify regions of alternative courses of events, i.e. return all hits from both queries. Example usage: `(or (atc C07.*) (atc C03.*))` matches patients using beta blockers (C07) and/or diuretics (C03).

Two operators can be used to constrain the matches returned by a query:

windowwith

Find all hits of a query not spanning more than a given time period. Example use: `(windowwith (atc C07.*) 364)` finds all periods where patients get prescriptions for a beta blocker that lasts for less than a year.

windowwithout

Find all windows of a given length in a history not matching a given query. Example: `(windowwithout (seq (test bp) (seq (test bp) (test bp))) 365)` finds a period without three successive blood pressure measurements within a year.

We have implemented one operator that calculates abstractions over intervals:

merge Find overlapping and adjacent hits from two queries and make one hit from all of these. An optional maximum spacing can be specified to introduce fuzziness of merging. This operation is useful in deducing medication intervals, i.e. periods when a patient is receiving a given drug, regardless of renewals of the prescription. The merge operation is then applied to all occurrences of a prescription of a given drug. Example: `(merge (atc C07.*) 7)` finds all periods the patient is medicated with a beta blocker, ignoring that the prescription has been renewed and allowing the patient to be one week late in getting a new prescription when the old one expires.

Two operators deal with degeneration of intervals to points:

startof/endof

Degenerate interval hits to points by returning their start- or endpoints. This is useful to reduce an `atc` query to the date of prescription. Example: `(startof (atc C07.*))` finds all dates where a beta blocker was prescribed.

Finally, we have implemented one operator that we believe is quite specific to our domain:

first Find the first match of a given query in the entire history. For example, the query `(seq (icpc K86) 0 7 (first (icpc K77)))` matches a diagnosis of hypertension followed by a diagnosis of infarction within a week, but only if the infarction is the patient's first ever.

In addition, the language defines shortcuts (“syntactic sugar”) for some usual con-

structs.

The syntactic sugar is defined as follows (`-->` is used to indicate “translates to”):

```
(prescription A)    --> (startof (atc A))
(medinterval A [I]) --> (merge (atc A) [I])
(medstart A [I])   --> (startof (medinterval A [I]))
(medend A [I])     --> (endof (medinterval A [I]))
(bp)               --> (test bp)
(diag I)           --> (icpc I)
```

These constructs work as follows:

prescription

The date when a given drug was prescribed.

medinterval

Overlapping and adjacent prescriptions, merged to form longer intervals of medication. This is more meaningful than looking at the individual prescription, since the latter is an administrative concept used to control medication usage.

medstart/medend

Start/end of the above medication period. This is useful for finding the date a patient starts or ends treatment with a given drug, as opposed to every renewal of the prescription.

bp

Simple alias for `(test bp)`, to save some typing.

diag

Simple alias for the diagnosis instance operator, using the term “diag” instead of “icpc”, for symmetry with the medication period operator.

EXAMPLE

One case found to be interesting during the case study, was the investigation of patients that are medicated before their blood pressure has been controlled at least three

times. This is against recommended practice, unless there are other indications that suggest medication (such as extremely high blood pressure).

In the query language, the above may be formulated as follows:

```
(seq (windowwithout (seq (bp) (seq (bp) (bp))) 365)
      0 0
      (first (medstart C0([2378]|9[ABCD]).*)))
```

This means that there should be a time window of one year (365 days) without three sequential blood pressure measurements followed by the first medication with a drug from the classes used for treating hypertension. Note that this query also extracts patients not suffering from hypertension (e.g. they have these medications prescribed for other reasons).

3.4.2 Algorithms

This section describes the algorithms used in implementation of the query language. It starts out describing the strategy of lazy execution and defining the terminology that is necessary, before it introduces the operators.

LAZY EXECUTION

³This is also called demand-driven computing.

Lazy execution³ is a pattern of implementation where the results are calculated when needed [vRHo4]. A lazy function typically returns its values one by one, always continuing calculation from the last result that was returned. The advantage is that if the caller only need one or a few results, there is no need to calculate the entire set of results. Also, this technique may also allow the programmer to calculate with infinite data structures, although this is not exploited in our implementation.

Our prototype employs lazy execution for extraction of matches from a query. We realise this through stateful objects that produce their results one by one as requested, with an optional time constraint (a match is required to be on or after the given time). In the description of the algorithms, a functional programming approach is taken to ease the discussion of the algorithms: All state information is passed as parameters

to the algorithms. Also, to simplify the presentation, no time constraints are passed down the tree on execution.

Figure 3.9 shows the operation of a search for a sequence “A, B” with the constraint that the gap between the “A” and the “B” should be between 3 and 9 units, inclusive. The illustration shows three entities in the query tree: The sequence operator (blue square) and two event matchers (red and green ovals). Along the bottom, the history is shown with time indicated below the timeline. The state of the event matchers is shown by the arrows pointing into the history. When the application requests an element (`nextMatch()`), the sequence operator instructs the “A” matcher to find the first hit. Then, the “B” matcher is advanced until a hit is found that satisfies the constraints of the sequence operator. As soon as this is returned, the compound hit is returned to the application. When the application requests the next element, the query continues from its current state, further advancing the “B” matcher until the upper time constraint is exceeded.

An algorithm using lazy execution is likely to perform fewer steps than its eager counterpart. However, the asymptotic analysis is identical, since in the worst case, the algorithm must calculate all results. The benefits from lazy execution is situation-dependent: In our setting, the top-level operations select and sort only use in the first hit from a query, and in this case it is likely that there are considerable savings in resource usage. In addition, the one- and first-operators effectively cut off branches in the query evaluation tree for additional savings. The size of these savings are, however, difficult to assess. Probably, we would have to run tests of “representative” queries (after having figured out what that would mean) on real data, and then compare these to an eager implementation of the same algorithms.

SYMBOLS AND DEFINITIONS

In the presentation of the algorithms, several definitions are needed. First, a *query* needs to be defined:

DEFINITION 3.1 (QUERY)

A query is an expression in the query language. It is written $Q(p_1, p_2, \dots, p_n)$, where p_i is parameter i .

Each query matches time intervals:

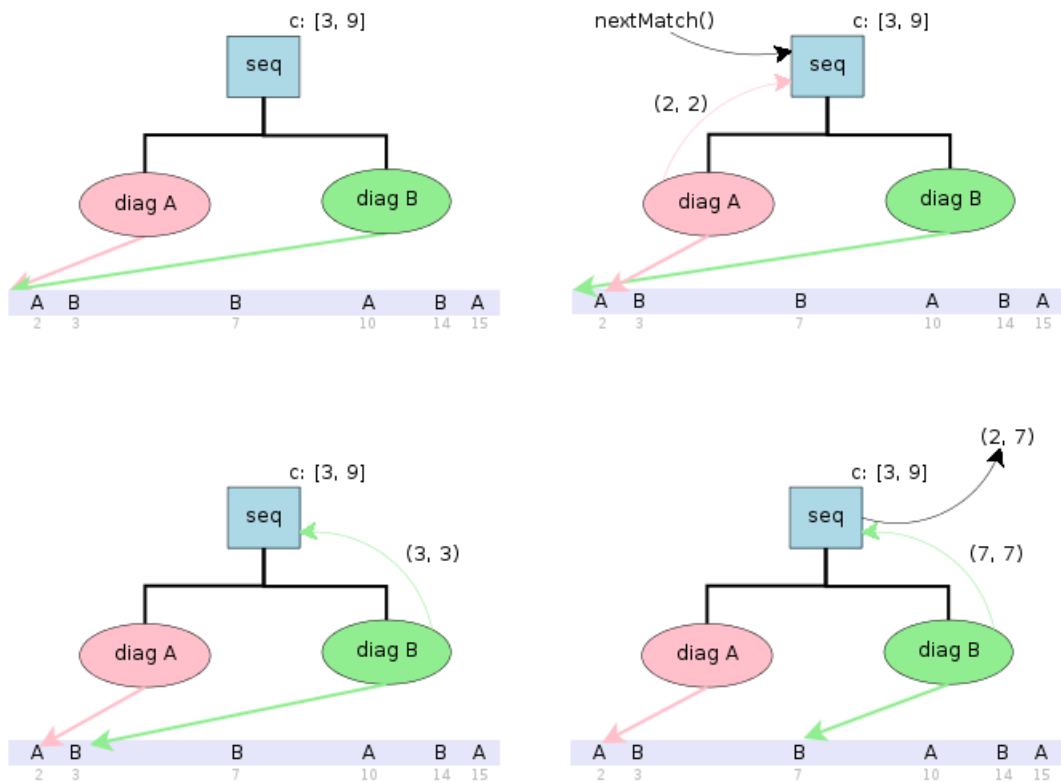


Figure 3.9: How the first result of a simple query is found: Upon request for a match, the request is passed down the query tree to the leaf nodes. In the example, the “A” iterator returns its first match, at time 2. Then, the “B” iterator returns matches after time 2 until a match satisfying the constraints of the seq is found – in this case the B at time 7. When the seq has found a match, this is immediately returned. If the caller requests another match, the iterators will continue from their current positions.

DEFINITION 3.2 (MATCH)
 A match is an ordered tuple (t_1, t_2) defining its start and end times. Operators that return point matches (single dates), return (t_1, t_2) where $t_1 = t_2$.

Two functions are defined on a match $m = (t_1, t_2)$:

$s(m)$ The function $s(m)$ returns the starting time of match m , i.e. $s(m) = t_1$.

$e(m)$ The function $e(m)$ returns the ending time of match m , i.e. $e(m) = t_2$.

Throughout this thesis, *match* and *hit* are used interchangeably. From the definitions of Query and Match, we are able to describe the result of a query:

DEFINITION 3.3 (QUERY RESULT)

A query result is a partially ordered set (poset) of matches resulting from the execution of a query on a given history, with the ordering of \leq on start time of each of its constituent hits.

The mapping of queries to results is defined by the function $R(Q, h)$, which is a function from the query, $Q(p_1, p_2, \dots, p_n)$, and the ordered history, h , to the poset of matches:

$$R(Q(p_1, p_2, \dots, p_n), h) : \{Q(p_1, p_2, \dots, p_n)\} \times (\{h\}, \leq) \rightarrow (\{(t_1, t_2)\}, \leq)$$

Set cardinality notation is used to denote the number of elements in a query result, i.e. $|r|$ denotes the number of elements in the result r .

With basis in the previous definitions, it is possible to define the semantics of a query as a function from the query and its parameters (other queries, constraints) to the query result of that query. This is done for each operator in the sections that follow. For brevity of notation, the notation for partially ordered sets is omitted when referring to query results in the following text.

In the algorithm listings, the following routines and symbols are used:

BinarySearch(M, t)

Return the index of the first match that starts after time t in the set M of matches. If no such match exists, return ∞ (which is an invalid index). The binary search algorithm is given in most algorithm textbooks, such as [CLRS01].

ExecuteQuery(Q, S, h)

Execute query Q over history h . The parameter S represents a tuple that is the state information needed to implement lazy execution. Implementations for each Q in the query language are given in the following sections.

Nothing The symbol “Nothing” is used to indicate the absence of a value.

Overlaps(a, b)

A function returning true if the two given matches overlap. If any of them is “Nothing”, the result is defined to be true.

When analysing the algorithms, the following symbols are used (these functions rely on the history h , but this is omitted for brevity of notation):

$S(Q)$ The space complexity of running EXECUTEQUERY(Q, \dots).

$T(Q)$ The time complexity of running EXECUTEQUERY(Q, \dots).

$N(Q)$ The maximum number of items returned by EXECUTEQUERY(Q, \dots).

INTRODUCTION TO THE ALGORITHMS

All algorithms are described with:

- Their *name*, specified as the first parameter to EXECUTEQUERY, which is a polymorphic function: Which version of the function is executed depends on which query is its first argument. For example, there are definitions for EXECUTEQUERY(Q_{or}, S, h) and EXECUTEQUERY(Q_{and}, S, h). When calling the function as EXECUTEQUERY $Q_{\text{and}}(Q_1, Q_2), S, h$ where Q_1 and Q_2 represent queries, the implementation of the and-operator is run.
- A *parameters definition*. This lists the parameters’ names and types. In particular, the state tuple used in the lazy execution is listed here. It is assumed a sort of pattern-matching, mapping the contents of the state parameter to its constituent variables; e.g. the state $S = \{a, b, c\}$ means that the function expects a parameter which is an ordered tuple of three variables, and these are referred to in the function body separately (a , b and c are referenced without other declaration than appearing as part of S).
- Their *return value*, in terms of what set the returned matches belong to. All algorithms return matches in the order defined by the ordering relation on the poset given by $R(Q, h)$, i.e. they return their matches sorted on starting time. This behaviour is relied upon in the implementations.

The sections that follow (except the two first, about the top-level extraction algorithm and the bottom-level search) are organised in three parts:

- *Introduction and definition:* An introductory paragraph informally describes the operator in question, and a definition in set notation, using the symbols described above, is given.
- *Algorithm:* A textual description of the algorithm explains an accompanying algorithm listing.
- *Asymptotic analyses:* Asymptotic running time- and space complexity is deduced. The number of elements that may be returned by a query is also given, since the other measures depend on this.

The algorithms to be described follow the same pattern of implementation:

- *Initialisation:* If the given state definition (always named S) does not exist, assign initial values to all variables of the state definition. All state variables have a short comment describing them.
- *Search:* For primitive operators: Traverse the history and find matching elements. For compound operators: Extract results from the sub-query/queries and combine them in the manner prescribed by the definition of the operator in question.
- *Termination:* Update state information to reflect progress of the search, and return a tuple with the match that was found in the first position and the (updated) state definition in the second. The former is “Nothing” when no further matches can be found.

TOP-LEVEL ALGORITHM: EXTRACTION OF MATCHES

The algorithm $\text{ALLMATCHES}(Q, h)$ returns all matches resulting from running query Q over history h . This algorithm hides the threading of states between invocations of EXECUTEQUERY , and it is given in algorithm listing 1. It also serves as an illustration of how EXECUTEQUERY is intended to be used.

This algorithm adds a constant amount to the space and running time complexities of the query to be executed.

Algorithm 1 Extraction of all matches

ALLMATCHES(Q, h)**Parameters:** Query Q ; history h .**Return:** The returned set contains all matches of query Q over history h .

```

1:  $R \leftarrow \emptyset \triangleleft$  Accumulated result of  $Q$ 
2:  $S \leftarrow \emptyset \triangleleft$  State of execution of  $Q$ 
3:  $m, S \leftarrow \text{EXECUTEQUERY}(Q, S, h)$ 
4: while  $m \neq \text{Nothing}$  do
5:    $R \leftarrow R \cup \{m\}$ 
6:    $m, S \leftarrow \text{EXECUTEQUERY}(Q, S, h)$ 
7: end while
8: RETURN  $R$ 

```

EVENT SEARCH

On the lowest level of the query tree, events are searched for in the histories. The algorithm is shown in algorithm listing 2. Here, a function E is defined as a function from the set of entries in histories to the set of matches. This function determines if a given entry is considered to be a match or not:

$$E(e) = \begin{cases} (t_1, t_2) & \text{if } e \text{ is considered to be a match} \\ \text{Nothing} & \text{otherwise} \end{cases}$$

ASYMPTOTIC ANALYSES This algorithm visits all elements of the underlying history once, and it may return up to all elements of the history:

$$N(Q_{\text{ev}}) = |h|$$

Its running time complexity is thus:

$$T(Q_{\text{ev}}) = O(|h|)$$

The space complexity is related to the storage of the variable i , which is constant:

$$S(Q_{\text{ev}}) = O(1)$$

Algorithm 2 Event search algorithm

EXECUTEQUERY($Q_{ev}(E), S, h$)**Parameters:** Event matching function E ; state information $S = \{i\}$;
history h .**Return:** A match is returned if and only if it belongs to the set: $\{m \mid E(e \in h) = m \wedge m \neq \text{Nothing}\}$

```

1: if  $S = \emptyset$  then  $\triangleleft$  First time called: Initialise
2:    $i \leftarrow 0$   $\triangleleft$  Progress in history  $h$ 
3: end if
4: while  $i < |h|$  do
5:    $m \leftarrow E(h[i])$ 
6:    $i \leftarrow i + 1$ 
7:   if  $m \neq \text{Nothing}$  then  $\triangleleft$  Check if a match was returned
8:     RETURN  $\{m, \{i\}\}$ 
9:   end if
10: end while
11: RETURN  $\{\text{Nothing}, \{i\}\}$ 

```

FIRST

Sometimes, it is useful to search for the *first occurrence* of a pattern in a history. The first-operator does this, returning only the first hit from the underlying query. We do not define this operator in set notation. Algorithm listing 3 shows the algorithm for extracting one hit, using an indicator variable as state information, which is set to one when the one hit has been returned.

Algorithm 3 First-query algorithm

EXECUTEQUERY($Q_{first}(Q), S, h$)**Parameters:** Query Q ; state information $S = \{x\}$; history h .**Return:** The first match returned by Q is returned once.

```

1:  $m \leftarrow \text{Nothing}$ 
2: if  $S \neq \emptyset$  then
3:    $m, S_Q \leftarrow \text{EXECUTEQUERY}(Q, \emptyset, h)$   $\triangleleft$  Get one match, ignore returned state
   value
4: end if
5: RETURN  $\{m, \{1\}\}$   $\triangleleft$  Return with a non- $\emptyset$  state to prevent execution next time

```

ASYMPTOTIC ANALYSES The first-operator returns a single hit, i.e.:

$$N(Q_{\text{first}}) = 1$$

It uses a single indicator variable as state information:

$$S(Q_{\text{first}}) = O(1)$$

When it comes to running-time complexity, this is difficult to estimate because of the lazy evaluation: While only extracting one hit, this may involve a lot of computation in the subtree represented by Q , but in general this will be much less than if all hits from Q were extracted. Nevertheless, an upper limit is:

$$T(Q_{\text{first}}) = T(Q)$$

DISJUNCTION

A disjunction (i.e. the or operator) finds alternative histories; it returns all matches from its two sub-queries.

DEFINITION 3.4 (DISJUNCTION QUERY)

The disjunction is defined over the sub-queries Q_1 and Q_2 as:

$$R(Q_{\text{or}}(Q_1, Q_2), h) = \{m \mid m \in R(Q_1, h) \cup R(Q_2, h)\}$$

Algorithm listing 4 shows the algorithm for finding a disjunction of the matches from two queries. The algorithm merges (as in the “merge sort” algorithm [CLRS01]) the matches of the two underlying queries.

ASYMPTOTIC ANALYSES This algorithm always returns all matches from both queries. This results in the following number of matches:

$$N(Q_{\text{or}}) = N(Q_1) + N(Q_2)$$

Each sub-query is executed exactly once, and the running time complexity of the algorithm is:

$$T(Q_{\text{or}}) = T(Q_1) + T(Q_2) + O(N(Q_1) + N(Q_2))$$

Algorithm 4 Disjunction query algorithmEXECUTEQUERY($Q_{\text{Or}}(Q_1, Q_2), S, h$)**Parameters:** Queries Q_1 and Q_2 ; state information $S = \{S_1, S_2, m_1, m_2\}$; history h .**Return:** A match is returned if and only if it belongs to the set: $R(Q_1, h) \cup R(Q_2, h)$

```

1: if  $S = \emptyset$  then  $\triangleleft$  First time called: Initialise
2:    $S_1 \leftarrow \emptyset$   $\triangleleft$  State of execution of  $Q_1$ 
3:    $S_2 \leftarrow \emptyset$   $\triangleleft$  State of execution of  $Q_2$ 
4:    $m_1, S_1 \leftarrow \text{EXECUTEQUERY}(Q_1, S_1, h)$ 
5:    $m_2, S_2 \leftarrow \text{EXECUTEQUERY}(Q_2, S_2, h)$ 
6: end if
7: if  $(m_1 = \text{Nothing}) \wedge (m_2 = \text{Nothing})$  then
8:   RETURN  $\{\text{Nothing}, \{S_1, S_2, m_1, m_2\}\}$ 
9: else if  $(m_1 = \text{Nothing}) \vee (m_2 \leq m_1)$  then
10:   $m \leftarrow m_2$ 
11:   $m_2, S_2 \leftarrow \text{EXECUTEQUERY}(Q_2, S_2, h)$ 
12:  RETURN  $\{m, \{S_1, S_2, m_1, m_2\}\}$ 
13: else  $\triangleleft$  Know that  $(m_1 \neq \text{Nothing}) \vee (m_1 < m_2)$ 
14:   $m \leftarrow m_1$ 
15:   $m_1, S_1 \leftarrow \text{EXECUTEQUERY}(Q_1, S_1, h)$ 
16:  RETURN  $\{m, \{S_1, S_2, m_1, m_2\}\}$ 
17: end if

```

Its space usage is a constant plus the space used by each sub-query:

$$S(Q_{\text{Or}}) = S(Q_1) + S(Q_2) + O(1)$$

WINDOW-WITH

The window-with operator returns all matches from a single query Q that do not span more than a maximum time w .

DEFINITION 3.5 (WINDOW-WITH QUERY)

The window with operator is defined as:

$$R(Q_{\text{Ww}}(Q, w), h) = \{m \mid (m \in R(Q, h)) \wedge (e(m) - s(m) \leq w)\}$$

Algorithm listing 5 shows the algorithm for finding hits within a given time window.

The algorithm checks each match of the sub-query and returns the matches whose extents are within the allowable window.

Algorithm 5 Window-with query algorithm

$$\text{EXECUTEQUERY}(Q_{\text{ww}}(Q_1, w), S, h)$$

Parameters: Query Q ; window size w ; state information $S = \{S_Q\}$;
history h .

Return: A match is returned if and only if it belongs to the set:

$$\{m \mid (m \in R(Q, h)) \wedge (e(m) - s(m) \leq w)\}$$

```

1: if  $S = \emptyset$  then  $\triangleleft$  First time called: Initialise
2:    $S_Q \leftarrow \emptyset$   $\triangleleft$  State of execution of  $Q$ 
3: end if
4: repeat
5:    $m, S_Q \leftarrow \text{EXECUTEQUERY}(Q, S_Q, h)$ 
6:   if  $(m \neq \text{Nothing}) \wedge (e(m) - s(m) \leq w)$  then
7:     RETURN  $\{m, \{S_Q\}\}$ 
8:   end if
9: until  $m = \text{Nothing}$ 
10: RETURN  $\{\text{Nothing}, \{S_Q\}\}$ 

```

ASYMPTOTIC ANALYSES In the worst case, all of the sub-query's hits are passed through, for a total number of matches of:

$$N(Q_{\text{ww}}) = N(Q)$$

The sub-query is executed exactly once, for a running time complexity of:

$$T(Q_{\text{ww}}) = T(Q) + O(N(Q))$$

Space usage is a constant, plus the space used by the sub-query:

$$S(Q_{\text{ww}}) = S(Q) + O(1)$$

MERGE

In the data model, medications are represented by a series of prescriptions of each drug. It may be interesting to extract the period in which a patient was treated with given medication, ignoring renewals of the prescription.

DEFINITION 3.6 (MERGE QUERY)

First, define a contiguous interval over a query with a maximum gap between adjacent intervals of δ :

$$D(Q, \delta, h) = \{(t_1, t_2) \mid ((m_1 \in D(Q, \delta, h)) \wedge (m_2 \in D(Q, \delta, h)) \wedge (s(m_2) - e(m_1) \leq \delta) \wedge (t_1 = s(m_1) \wedge t_2 = e(m_2))) \vee ((t_1, t_2) \in R(Q, h)))\}$$

Then, the merge operator is defined to return the largest contiguous intervals:

$$R(Q_{\text{merge}}(Q, \delta), h) = \{(t_1, t_2) \mid (t_1, t_2) \in D(Q, \delta, h) \wedge \neg \exists (u_1, u_2) \in D(Q, \delta, h). (((u_1 < t_1) \wedge (u_2 - t_1 \geq \delta)) \vee ((u_1 - t_2 \leq \delta) \wedge (u_2 > t_2)))\}$$

Algorithm listing 6 shows the algorithm for finding the largest merged intervals. Since the queries are defined to return their hits sorted on start time, the abstraction amounts to scanning the hits from the underlying query sequentially, merging them to a single match until the gap between the latest end time seen so far and the start time found is larger than δ .

ASYMPTOTIC ANALYSES In the worst case, all of the sub-query's hits are passed through separately, for a total number of matches of:

$$N(Q_{\text{merge}}) = N(Q)$$

The sub-query is executed exactly once, for a running time complexity of the algorithm of:

$$T(Q_{\text{merge}}) = T(Q) + O(N(Q))$$

Its space usage is a constant plus the space used by the sub-query:

$$S(Q_{\text{merge}}) = S(Q) + O(1)$$

Algorithm 6 Merge query algorithm

EXECUTEQUERY($Q_{\text{merge}}(Q, w), S, h$)**Parameters:** Query Q ; tolerance δ ; state information $S = \{S_Q, \alpha\}$; history h .**Return:** A match is returned if and only if it belongs to the set:

$$R(Q_{\text{merge}}(Q, \delta), h) = \{(t_1, t_2) \mid (t_1, t_2) \in D(Q, \delta, h) \wedge \\ \neg \exists (u_1, u_2) \in D(Q, \delta, h). (((u_1 < t_1) \wedge (u_2 - t_1 \geq \delta)) \vee \\ ((u_1 - t_2 \leq \delta) \wedge (u_2 > t_2)))\}$$

```

1: if  $S = \emptyset$  then  $\triangleleft$  First time called: Initialise
2:    $S_Q \leftarrow \emptyset$   $\triangleleft$  State of execution of  $Q$ 
3:    $\alpha \leftarrow \text{Nothing}$   $\triangleleft$  Match that is under construction
4: end if
5:  $m, S_Q \leftarrow \text{EXECUTEQUERY}(Q, S_Q, h)$ 
6: while  $m \neq \text{Nothing}$  do
7:   if  $\alpha = \text{Nothing}$  then
8:      $\alpha \leftarrow m$ 
9:   else
10:    if  $s(m) - e(\alpha) \leq \delta$  then  $\triangleleft$  Can extend  $\alpha$ 
11:       $\alpha \leftarrow (s(\alpha), \max(e(\alpha), e(m)))$ 
12:    else  $\triangleleft$  Reset and return  $\alpha$ , using  $m$  for  $\alpha$  next time
13:      RETURN  $\{\alpha, \{S_Q, m\}\}$ 
14:    end if
15:  end if
16:   $m, S_Q \leftarrow \text{EXECUTEQUERY}(Q, S_Q, h)$ 
17: end while
18: if  $\alpha \neq \text{Nothing}$  then  $\triangleleft$  Out of matches, return what we have
19:   RETURN  $\{\alpha, \{S_Q, \text{Nothing}\}\}$ 
20: end if
21: RETURN  $\{\text{Nothing}, \{S_Q, \alpha\}\}$ 

```

AVOIDING RE-RUNNING A QUERY: A CACHING PROXY

To avoid re-evaluation of a subtree of the query tree whenever an algorithm needs to use the results of the subtree more than once, a caching proxy⁴ function for EXECUTE-QUERY is employed. This function, GETCACHED, returns all hits from a query, but during the first access, the hits are cached. Regardless of which elements are sought, all elements are stored for later. When the calling function seeks an element that has already been cached, the proxy looks this up in the cache, avoiding to re-run the subtree represented by the cached query. The proxy can find a new starting point for sequential return of hits by using binary search. This behaviour is triggered by passing a parameter t that is different from “Nothing”, to the function. If $t = \text{Nothing}$, the matches are returned sequentially from the last match returned. The algorithm is shown in algorithm listing 7.

The main use of this function is in the sequence operator. Here, the results from a query Q_2 is combined with all hits from query Q_1 . In this case, caching the hits from Q_2 avoids running the subtree represented by Q_2 more than once. In addition, the operator specifies that the hits from Q_2 should start after the end of the hit from Q_1 . This means that not all hits from Q_2 need to be processed for every hit from Q_1 , hence the binary search to find a new starting point.

ASYMPTOTIC ANALYSES Considering the running time complexity, two cases must be considered: If the elements that are searched have been extracted from the underlying query, or not. Furthermore, the running time differs between sequential return of the elements, and search for a place to start.

If the elements have not been extracted, the matches are sequentially retrieved from the underlying query, up to the point in time that is requested, yielding a time complexity of $T(Q) + O(N(Q))$. This is the case both when returning elements sequentially and when seeking a starting point.

In the case of the presence of the elements in the cache (at least up to the requested time, if any), the complexity depends on the value of t : If $t \neq \text{Nothing}$, the time to find the nearest element is $O(\log_2 |C|)$. Else, the lookup time is constant, $\theta(1)$, assuming that the cache is random-access with uniform access time.

Considering space complexity, this is related to the size of the cache, $|C|$, the space requirements of the underlying query, $S(Q)$, plus a constant for the storage of local variables and i . In sum: $S(Q) + O(|C|)$. Assessing the size of the cache is difficult

⁴We call this a proxy, as it is a functional-programming equivalent of a *virtual proxy* in the “Proxy” pattern described in books on design patterns (such as [Co098]). A virtual proxy defers loading of a large or complex object until it is needed.

Algorithm 7 Caching proxy for ExecuteQuery

GETCACHED(Q, t, S, h)**Parameters:** Query Q ; optional time t ; state information $S = \{C, S_Q, i\}$; history h .**Return:** The returned matches are all matches of query Q over history h .

```

1: if  $S = \emptyset$  then  $\triangleleft$  First time called: Initialise
2:    $C \leftarrow \emptyset$   $\triangleleft$  Cache
3:    $S_Q \leftarrow \emptyset$   $\triangleleft$  State of execution of  $Q$ 
4:    $i \leftarrow 0$   $\triangleleft$  Index into  $C$ 
5: end if
6: if  $(t \neq \text{Nothing}) \wedge (C \neq \emptyset) \wedge (t \leq s(C[|C| - 1]))$  then  $\triangleleft$  Find new index
7:    $i \leftarrow \text{BINARYSEARCH}(C, t)$ 
8: else if  $i \geq |C|$  then
9:   repeat  $\triangleleft$  Linear search and cache construction/expansion
10:     $m, S_Q \leftarrow \text{EXECUTEQUERY}(Q, S_Q, h)$ 
11:    if  $m = \text{Nothing}$  then
12:      RETURN  $\{\text{Nothing}, \{C, S_Q, i\}\}$ 
13:    end if
14:     $C \leftarrow C \cup \{m\}$   $\triangleleft$  Extend cache
15:  until  $(t \neq \text{Nothing}) \rightarrow (s(m) \geq t)$ 
16:   $i \leftarrow |C| - 1$ 
17: end if
18: RETURN  $\{C[i], \{C, S_Q, i + 1\}\}$ 

```

because of lazy evaluation, but an upper bound is the number of elements returned by the underlying query: $O(N(Q))$.

The overall space and running time characteristics of the function will depend on the usage pattern: Because of lazy evaluation, it may very well be the case that not all of the underlying matches are extracted and cached, and the use of binary search may vary according to the calling function's requirements.

SEQUENCE

A sequence query finds matches from two queries, where the matches from the second query or start on or after the end of matches from the first. The resulting match from the sequence query spans the extent of both matches. There may be a constraint on the gap between the end of the first match and the start of the second. Consider-

ing this constraint involves deciding on the semantics when the gap between events can be zero, i.e. whether the operator accepts events that happened on the same day⁵ as being “sequential”. We define the sequence operator to behave this way, implying the use of ‘≤’ instead of ‘<’ in the definition below. Allowing c_2 to be zero, extends the possibilities: If $c_1 = c_2 = 0$, this makes the operator specify simultaneousness of events. However, it must be a requirement that $c_1 \leq c_2$.

⁵Days is the unit of resolution of the time representation in the system, because the database is not accurate at lower granularities.

DEFINITION 3.7 (SEQUENCE QUERY)

The sequence query is defined as:

$$R(Q_{\text{seq}}(Q_1, Q_2, c_1, c_2), h) = \{(s(m_1), e(m_2)) \mid \\ ((m_1, m_2) \in R(Q_1, h) \times R(Q_2, h)) \wedge \\ (0 \leq c_1 \leq (s(m_2) - e(m_1)) \leq c_2)\}$$

Here, the following symbols are used:

Q_1, Q_2 Sub-queries.

c_1, c_2 Constraints on the spacing between elements of the sequence.

Algorithm listing 8 shows the algorithm for finding sequences. The algorithm returns for each match in the first query, Q_1 , all matches from the second query, Q_2 , so that the gap between the end of the first and start of the second is within the interval of the constraints. Figure 3.9 shows an example of how the sequence search operates.

ASYMPTOTIC ANALYSES The worst-case complexity for this algorithm is exposed when all hits from Q_1 end prior to any hit from Q_2 and the constraining window is large enough to encompass any gap between the end of a hit from Q_1 and the start of a hit from Q_2 . In this case, all hits from Q_2 are returned for each hit of Q_1 , for a total number of elements:

$$N(Q_{\text{seq}}) = N(Q_1) \cdot N(Q_2)$$

Since the algorithm caches hits from Q_2 , each sub-query is executed only once. The caching proxy induces a binary search for each hit of Q_1 , and the total running time complexity for the sequence query algorithm is thus:

$$T(Q_{\text{seq}}) = T(Q_1) + T(Q_2) + O(N(Q_1)(N(Q_2) + \log_2 N(Q_2))) \\ = T(Q_1) + T(Q_2) + O(N(Q_1) \cdot N(Q_2))$$

Algorithm 8 Sequence query algorithmEXECUTEQUERY($Q_{seq}(Q_1, Q_2, c_1, c_2), S, h$)**Parameters:** Queries Q_1 and Q_2 ; constraining times c_1 and c_2 ; state information $S = \{m_1, S_1, C_2\}$; history h .**Return:** A match is returned if and only if it belongs to the set:

$$R(Q_{seq}(Q_1, Q_2, c_1, c_2), h) = \{(s(m_1), e(m_2)) \mid \\ ((m_1, m_2) \in R(Q_1, h) \times R(Q_2, h)) \wedge \\ (0 \leq c_1 \leq (s(m_2) - e(m_1)) \leq c_2)\}$$

```

1: if  $S = \emptyset$  then  $\triangleleft$  First time called: Initialise
2:    $m_1 \leftarrow \text{Nothing}$   $\triangleleft$  Current match from  $Q_1$ 
3:    $S_1 \leftarrow \emptyset$   $\triangleleft$  State of execution of  $Q_1$ 
4:    $C_2 \leftarrow \emptyset$   $\triangleleft$  State of GetCache over  $Q_2$ 
5: end if
6: loop  $\triangleleft$  For all matches from  $Q_1$ 
7:    $t \leftarrow \text{Nothing}$ 
8:   if  $m_1 = \text{Nothing}$  then  $\triangleleft$  Needs to find a new match from  $Q_1$ 
9:      $m_1, S_1 \leftarrow \text{EXECUTEQUERY}(Q_1, S_1, h)$ 
10:    if  $m_1 = \text{Nothing}$  then  $\triangleleft$  Termination: All of  $Q_1$  exhausted
11:      RETURN  $\{\text{Nothing}, \{m_1, S_1, C_2\}\}$ 
12:    end if
13:     $t \leftarrow e(m_1) + c_1$   $\triangleleft$  Search from this time
14:  end if
15:  loop  $\triangleleft$  Try all matches in  $Q_2$  against constraints
16:     $m_2, C_2 \leftarrow \text{GETCACHED}(Q_2, t, C_2, h)$ 
17:    if  $(m_2 = \text{Nothing}) \vee (s(m_2) > e(m_1) + c_2)$  then  $\triangleleft$  No hit or out of range
18:       $m_1 \leftarrow \text{Nothing}$ 
19:      BREAK  $\triangleleft$  Break out of innermost loop
20:    else if  $m_2 \neq m_1$  then  $\triangleleft$  An event does not follow itself
21:      RETURN  $\{(s(m_1), e(m_2)), \{m_1, S_1, C_2\}\}$ 
22:    end if
23:  end loop
24: end loop

```

The space complexity is related to the cache of results for Q_2 plus the space complexity of the two queries:

$$S(Q_{seq}) = S(Q_1) + S(Q_2) + O(N(Q_2))$$

CONJUNCTION

A conjunction (i.e. the and operator) finds parallel patterns; the cartesian product of matches from two queries. The resulting matches span from the start of the match from the first query to the end of the match from the second. No parameters are given apart from the sub-queries.

DEFINITION 3.8 (CONJUNCTION QUERY)

The conjunction is defined over the sub-queries Q_1 and Q_2 as:

$$R(Q_{\text{and}}(Q_1, Q_2), h) = \{(s(m_1), e(m_2)) \mid (m_1, m_2) \in R(Q_1, h) \times R(Q_2, h)\}$$

Algorithm listing 9 shows the algorithm for finding a conjunction, or cartesian product, of the matches from two queries. The algorithm returns for each match in the first query, Q_1 , all matches from the second query, Q_2 .

ASYMPTOTIC ANALYSES This algorithm always return all hits from Q_2 for each hit of Q_1 , for a total number of returned elements of:

$$N(Q_{\text{and}}) = N(Q_1) \cdot N(Q_2)$$

Basically, the algorithm is the same as the sequence query algorithm, except for the constraints (both being based on the cartesian product), giving the same asymptotic running time:

$$T(Q_{\text{and}}) = T(Q_1) + T(Q_2) + O(N(Q_1) \cdot N(Q_2))$$

The last term would include a logarithmic component from the binary search in `GETCACHED`. This may be avoided, for instance by adding a special case to `GETCACHED` that starts from the beginning of the cache if $t = -\infty$. Another possibility is to build the caching into the implementation of the and operator.

Also, the space complexity is the same as for the sequence operator:

$$S(Q_{\text{and}}) = S(Q_1) + S(Q_2) + O(N(Q_2))$$

Algorithm 9 Conjunction query algorithm

EXECUTEQUERY($Q_{\text{and}}(Q_1, Q_2), S, h$)**Parameters:** Queries Q_1 and Q_2 ; state information $S = \{m_1, S_1, C_2\}$;
history h .**Return:** A match is returned if and only if it belongs to the set: $\{(s(m_1), e(m_2)) \mid (m_1, m_2) \in R(Q_1, h) \times R(Q_2, h)\}$

```

1: if  $S = \emptyset$  then  $\triangleleft$  First time called: Initialise
2:    $m_1 \leftarrow \text{Nothing}$   $\triangleleft$  Current match from  $Q_1$ 
3:    $S_1 \leftarrow \emptyset$   $\triangleleft$  State of execution of  $Q_1$ 
4:    $C_2 \leftarrow \emptyset$   $\triangleleft$  State of GetCache over  $Q_2$ 
5: end if
6: loop  $\triangleleft$  For all matches from  $Q_1$ 
7:    $t \leftarrow \text{Nothing}$ 
8:   if  $m_1 = \text{Nothing}$  then  $\triangleleft$  Needs to find a new match from  $Q_1$ 
9:      $m_1, S_1 \leftarrow \text{EXECUTEQUERY}(Q_1, S_1, h)$ 
10:    if  $m_1 = \text{Nothing}$  then  $\triangleleft$  Termination: All of  $Q_1$  exhausted
11:      RETURN  $\{\text{Nothing}, \{m_1, S_1, C_2\}\}$ 
12:    end if
13:     $t \leftarrow -\infty$   $\triangleleft$  Start over enumerating  $Q_2$ 
14:  end if
15:  loop
16:     $m_2 \leftarrow \text{GETCACHED}(Q_2, t, C_2, h)$ 
17:    if  $m_2 = \text{Nothing}$  then  $\triangleleft$  No hit
18:       $m_1 \leftarrow \text{Nothing}$ 
19:      BREAK  $\triangleleft$  Break out of innermost loop
20:    else
21:      RETURN  $\{(s(m_1), e(m_2)), \{m_1, S_1, C_2\}\}$ 
22:    end if
23:  end loop
24: end loop

```

WINDOW-WITHOUT

The window-without operator finds periods of a specified length that do not contain any matches of a given query. This window may start at a date containing a match, but not end at such a date.

DEFINITION 3.9 (WINDOW-WITHOUT QUERY)

Formally, the window-without operator is defined as:

$$R(Q_{\text{Wwo}}(Q, w), h) = \{(x, y) \mid \neg \exists (m \in R(Q)). ((s(m) < y) \wedge (x < e(m))) \vee ((x < e(m)) \wedge (s(m) < y)) \wedge (y - x = w) \wedge (x, y \in \{\dots, -\delta, 0, \delta, 2\delta, \dots\}) \wedge (s(h[0]) \leq x \leq y \leq s(h[|h| - 1]))\}$$

Here, δ represent the granularity of the time representation (one day in the prototype).

In algorithm listing 10, the algorithm for the window-without operator is given. The algorithm begins at the start of the history and generates windows every δ time units until the window meets a match. When this happens, the start of the window is moved to the end of the match and the search continues. In the listing, the function $\text{OVERLAPS}(m_1, m_2)$ is true if the intervals defined by m_1 and m_2 overlap. An interval which is “Nothing” does not overlap any interval.

ASYMPTOTIC ANALYSES For each window between matches m_1 and m_2 with width $\Delta m = s(m_2) - e(m_1) \geq w$, the algorithm returns $\frac{\Delta m - w}{\delta} + 1$ matches. The time complexity is difficult to calculate exactly, but it will be dependent on $T(Q)$ (finding matches of Q), the size of the time step, δ , and the data’s characteristics.

One limiting case is the case where no matches of Q exist in the history. In this case, the number of windows returned equals the ratio of the length in time of the history and the time granularity:

$$N(Q_{\text{Wwo}}) = \left\lfloor \frac{s(h[|h| - 1]) - s(h[0]) - w}{\delta} \right\rfloor + 1$$

An upper bound on the running time is thus (see further discussion in section 5.3.3):

$$T(Q_{\text{Wwo}}) = T(Q) + O\left(\left\lfloor \frac{s(h[|h| - 1]) - s(h[0]) - w}{\delta} \right\rfloor\right)$$

Algorithm 10 Window-without query algorithm

EXECUTEQUERY($Q_{\text{Wwo}}(E), S, h$)**Parameters:** Query Q ; window size w ; state information $S = \{t, S_Q, m\}$;
history h **Return:** A match is returned if and only if it belongs to the set:

$$R(Q_{\text{Wwo}}(Q, w), h) = \{(x, y) \mid \neg \exists (m \in R(Q)). ((s(m) < y) \wedge (x < e(m))) \vee ((x < e(m)) \wedge (s(m) < y))) \wedge (y - x = w) \wedge (x, y \in \{\dots, -\delta, 0, \delta, 2\delta, \dots\}) \wedge (s(h[0]) \leq x \leq y \leq s(h[|h| - 1]))\}$$

```

1: if  $S = \emptyset$  then  $\triangleleft$  First time called: Initialise
2:    $t \leftarrow s(h[0])$   $\triangleleft$  Start of window to be returned
3:    $S_Q \leftarrow \emptyset$   $\triangleleft$  State of execution of  $Q$ 
4:    $m, S_Q \leftarrow \text{EXECUTEQUERY}(Q, S_Q, h)$   $\triangleleft$  Match we are checking against
5: end if
6: while  $t < s(h[|h|] - 1) - w$  do  $\triangleleft$  Within the bounds of the history
7:   if  $\neg \text{OVERLAPS}((t, t + w), m)$  then
8:      $m_r \leftarrow (t, t + w)$ 
9:      $t \leftarrow t + \delta$ 
10:    RETURN  $\{m_r, \{t, S_Q, m\}\}$ 
11:   else if  $m \neq \text{Nothing}$  then
12:      $m, S_Q \leftarrow \text{EXECUTEQUERY}(Q, S_Q, h)$ 
13:      $t \leftarrow e(m)$ 
14:   else
15:      $t \leftarrow t + \delta$ 
16:   end if
17: end while
18: RETURN  $\{\text{Nothing}, \{t, S_Q, m\}\}$ 

```

Concerning space complexity, the algorithm does no caching and thus only adds a constant amount of space to the total complexity of the combined query:

$$S(Q_{\text{ww0}}) = S(Q) + O(1)$$

SUMMARY

Table 3.1 shows the maximum number of elements generated by the different algorithms, in terms of the number of elements returned by their sub-queries. This table is needed when using table 3.2, showing algorithm time and space complexities, to calculate the complexity of executing a given query.

For example, consider the query:

```
(seq (windowwith (seq (bp) (seq (bp) (bp))) 365)
  0 0
  (first (medstart C0([2378]|9[ABCD]).*)))
```

This query could be expressed as:

$$Q = Q_{\text{seq}}(Q_{\text{ww}}(Q_{\text{seq}}(Q_{\text{ev}}(e_{\text{bp}}), Q_{\text{seq}}(Q_{\text{ev}}(e_{\text{bp}}), Q_{\text{ev}}(e_{\text{bp}}))), 365), \\ Q_{\text{first}}(Q_{\text{startof}}(Q_{\text{merge}}(Q_{\text{ev}}(e_{\text{atc}}))), \\ 0, 0)$$

where $e_{\text{bp}}(x)$ is a function returning a match (t, t) if x is a blood pressure measurement at time t . Correspondingly, $e_{\text{atc}}(y)$ returns a match (t_1, t_2) for a given prescription y .

To find the asymptotic running time of this query, expressions from the tables 3.1 and 3.2 are applied to the outermost operator (seq):

$$T(Q) = T(Q_{\text{ww}}(\dots)) + T(Q_{\text{first}}(\dots)) + O(N(Q_{\text{ww}}(\dots)) \cdot N(Q_{\text{first}}(\dots)))$$

Continuing to apply expressions from the tables to the formula above yields a total asymptotic complexity of $O(|h|^3)$.

From the example above, and from looking at the tables, it is possible to discuss the complexities of different queries. If we ignore the window-without operator, which

is discussed in greater detail in section 5.3.3, the time complexities listed in table 3.2 fall into two categories: Linear time (event search, disjunction, window-with, first, merge) and multiplicative time (conjunction, sequence). Building a tree from linear-time operations yields a total linear-time complexity. On the other hand, building a tree from operators multiplying their sub-queries' complexity functions results in a total complexity exponential in the height of the query tree. A real-world query will fall somewhere between these two extremes.

Query	Symbol(Q)	N(Q)
Event search	Q_{ev}	$ h $
And	Q_{and}	$N(Q_1) \cdot N(Q_2)$
Or	Q_{or}	$N(Q_1) + N(Q_2)$
Sequence	Q_{seq}	$N(Q_1) \cdot N(Q_2)$
Window-with	Q_{ww}	$N(Q)$
Window-without	Q_{wwo}	$\left\lfloor \frac{s(h[h -1]) - s(h[0]) - w}{\delta} \right\rfloor + 1$
First	Q_{first}	1
Merge	Q_{merge}	$N(Q)$

Table 3.1: Number of matches produced by the algorithms

Q	T(Q)	S(Q)
Q_{ev}	$O(h)$	$O(1)$
Q_{and}	$T(Q_1) + T(Q_2) + O(N(Q_1) \cdot N(Q_2))$	$S(Q_1) + S(Q_2) + O(N(Q_2))$
Q_{or}	$T(Q_1) + T(Q_2) + O(N(Q_1) + N(Q_2))$	$S(Q_1) + S(Q_2) + O(1)$
Q_{seq}	$T(Q_1) + T(Q_2) + O(N(Q_1) \cdot N(Q_2))$	$S(Q_1) + S(Q_2) + O(N(Q_2))$
Q_{ww}	$T(Q) + O(N(Q))$	$S(Q) + O(1)$
Q_{wwo}	$T(Q) + O\left(\left\lfloor \frac{s(h[h -1]) - s(h[0]) - w}{\delta} \right\rfloor\right)$	$S(Q) + O(1)$
Q_{first}	$T(Q)$	$O(1)$
Q_{merge}	$T(Q) + O(N(Q))$	$S(Q) + O(1)$

Table 3.2: Algorithm complexities

Exactly what the complexity of a real-world query will be is difficult to predict, but we can consider the theoretical extremes. Combining cartesian products will generate a huge amount of matches, exponential in the height of the query tree (i.e. number of nested operators). Searching in a collection of 778 hypertension histories yielded the results shown in table 3.3. Here, the “—” means that the computer ran out of memory. First, a test was run using a symmetrical tree of and-operators, and then the experiments was repeated using a sequence. Both produced a large number of results, even for fairly low trees. Comparing this to one of the linear-time operators,

“or”, shows that the difference is significant: Neither the number of matches nor the execution time is anywhere near that of the operators based on cartesian products.

Query	Results	Time
(bp)	2,642	0.081
(and (bp) (bp))	40,248	0.166
(and (and (bp) (bp)) (and (bp) (bp)))	—	—
(seq (bp) (bp))	21,454	0.149
(seq (seq (bp) (bp)) (seq (bp) (bp)))	5,973,444	11.067
(or (bp) (bp))	5,284	0.143
(or (or (bp) (bp)) (or (bp) (bp)))	10,568	0.283
(or (or (or ...)))	21,136	0.553
(or (or (or (or ...))))	42,272	1.156
(or (or (or (or (or ...))))))	84,544	1.902
(or (or (or (or (or (or ...))))))	169,088	3.554

Table 3.3: Running times for example queries. “—” means that the computer went out of memory. The first part of the table indicates the exponential nature of combining cartesian products in a tree. The second part shows that or-operator does not have the same problem.

These discussions are, however, theoretical. In our case study, all queries seemed to execute instantly. To determine whether complexity really is a problem, relevant queries must be gathered⁶ and their corresponding query trees analysed. If this analysis shows that the current specification is infeasible, the operators defined through the cartesian product need to be redefined.

⁶This is work that has commenced, but not as part of this thesis.



Case study

(chapter four)

Our experience from previous projects indicates that prototyping is a good approach for getting feedback in our problem area. We chose to develop our solution as described in the previous chapter, and then test it with a general practitioner. This helps us to gain a better understanding of the problem, and to identify weak or missing functionality in our design. Although the study recounted in this chapter is not sufficient as a basis for drawing firm conclusions about the success of our design, we hope that the experiences will build a better foundation for performing more thorough evaluations at a later stage.

Throughout this chapter, the following topics are addressed:

1. *Goals*: What we were looking for during the case study.
2. *Performing the study*: How we proceeded during the study.
3. *Findings*: Observations made during the sessions with the general practitioner.
4. *Summary of findings related to goals*

4.1 Goals

We decided upon the following goals, supporting the higher-order goal of investigating the potential of our visualisation design:

- Find possible uses of our visualisation design.
- Investigate the usefulness of the interaction features of the prototype.
- Look for limitations and problems with, as well as possibilities for improvements of, the prototype.

In the planning of such an experiment, one issue that needs to be addressed is its scope: The of the number of cases to investigate and the number of persons involved. We considered a small-scale study with a single general practitioner to be the best option at this stage, because this would allow us to assess if our design at all is useful and identify its most obvious shortcomings before embarking on larger studies. Section 6.3 lists the preliminary plan for continued research, where the study referred here is followed by several iterations of improvement and evaluation. The experiences from this case study will be important for design of the further evaluation, and for improving the prototype before the actual evaluation process starts.

4.2 Performing the study

We conducted the study by letting a general practitioner explore a group of patients from his own practice and comment on his findings, both of expected and unexpected cases. In advance, we had decided to investigate patients suffering from hypertension (see section 1.3). We consulted medical guidelines, such as the Norwegian Electronic Handbook for Physicians (NEL) [ASo6], and selected diagnosis codes and medications that were relevant to our problem. These were discussed with the physician, and the visualisation was parameterised accordingly.

Three persons were present during the experiment:

- General practitioner and leader of research at the Norwegian Centre of Electronic Health Records Research (NSEP), Anders Grimsmo (secondary tutor of this thesis).

- PhD student at the Institute of Computer and Information Sciences (IDI), Ole Edsberg (main tutor of this thesis).
- Master student at IDI, Stein Jakob Nordbø (author of this thesis).

The study was performed in four sessions spanning three days:

1. *Introduction*: The visualisation concept and prototype were introduced, and interesting problems were proposed. Also, missing features were spotted. After this session, the new features were implemented, and queries were constructed for the most interesting problems that were proposed.
2. *Query 1*: Investigation of the case “Patients being medicated for hypertension without three preceding controls of blood pressure.” The guideline states that there should be at least three controls before drugs are prescribed, as a main rule.
3. *Query 2*: Investigation of the case “Patients prescribed ACE-inhibitors without any known indications of use of this kind of drug.” ACE-inhibitors represent a costly form of treatment that should be avoided if there are other alternatives.
4. *Query 3*: Investigation of the case “Patients with diagnoses known as complications of hypertension, where the first such diagnosis occurs after the first hypertension diagnosis.” These patients were sorted on the time from the first diagnosis of hypertension to the first complication.

4.2.1 Limitations

Besides measuring the effectiveness of the visualisation design, an evaluation of an interactive prototype measures the interaction capabilities, since interaction is necessary to explore the data. This is not necessarily a disadvantage – the interaction mechanisms are important – but it can be difficult to tell if more insight into the data could be generated with a different interaction design. Another uncertainty factor would be whether the interaction mechanisms or the visualisation itself accounts the most for the observations.

A problem with the evaluation being affected by the user interface is the lack of a user-friendly interface in certain areas of our prototype. To minimise the impact of this, the physician was assisted in carrying out tasks for which no user-friendly user interface had been implemented: Loading data from the database and specification and execution of queries. Thus, the physician ended up scrolling the visualisation, inspecting it using the brushing (“mouse-over”) facilities, and changing between the

visualisation and the record view. Another limitation stem from our helpfulness: It is conceivable that the user would be less inclined to make use of the operations that he was not able to carry out himself. The general practitioner actually stated that he would like to try out the tool for himself sometime.

In addition, an evaluation of a visualisation will be influenced by the data that is inspected: At least, the data must contain some interesting information to discover, and the user must possess the domain knowledge necessary to identify it when it appears. We try to remedy this by choosing a test user that knows the data and has long experience in the field of general practice medicine.

A third factor influencing this study is the motivation of its participants. Our test subject is leader of research at a centre for electronic health records research and is interested in technology. Besides, he participated as a co-author of the scientific article resulting from this study (appendix D). For these reasons, we believe to have found a motivated user. On the other hand, it is conceivable that the test subject can have a certain bias.

4.3 Findings

This section lists findings from the four sessions outlined above. See section 5.1.1 for a discussion of these results.

4.3.1 Introductory session

In the introductory session, the visualisation and prototype were introduced through a brief demonstration. Since we had prepared the visualisation for hypertension patients, we extracted this group from the database, and the align operation was used to bring the first occurrence of a hypertension diagnosis for all the patients together. Then, we asked the general practitioner to come up with a relevant problem. This resulted in the formulation of the queries discussed in the following sections. During the study, help was given to the physician when requested.

FEATURE REQUESTS

During this session, the need for the following features was discovered:

- *Improved accuracy in finding blood pressure measurements:* Blood pressure measurements are extracted from the free-text contact notes using a regular expression. The original expression found all measurements on the form “BT: xxx/yyy”, but many measurements were coded on different forms, for example: “...measured the BT to be xxx/yyy”. This needed to be remedied to improve the quality of the query results, as one of the queries specifically searched for patients without a given number of blood pressure measurements.
- *Faster access to the contact notes:* The journal notes are vital in understanding the reasons for coding diagnoses and prescribing drugs: They give the grounds for the decision from a medical point of view or through additional information (such as a change of medications because the patient suffered from side-effects). Because of this, the general practitioner understood many cases simply by looking at the relevant contact note. Since he opened and closed the record viewer window all the time, a panel showing the contact note under the mouse pointer was proposed.
- *Highlighting individual patients:* Sometimes, certain indications might explain treatment patterns, and making it easy to spot patients exhibiting these indications increases the value of the visualisation. In our study, this was discovered when the general practitioner stated that diabetes patients often were treated differently. This was proposed to be implemented using a colouring of those patients’ id-numbers on the y-axis.

After this session, the proposed features were implemented, and the agreed-upon queries were prepared.

4.3.2 Query 1: Starting medication

The second session was devoted to exploring the case:

“Patients being medicated for hypertension without three preceding controls of blood pressure”

This was considered an interesting case because it is against recommended practise, and a surprisingly high number of patients with this violation were found during

the first inspection of the data. After the first session, the extraction of blood pressure measurements was improved, so that in the second session, it turned out to be 35 relevant cases. We discovered that many of these did not actually suffer from hypertension (hence the missing measurements), and after adding the requirement that the patients should have been given the hypertension diagnosis, only five cases were left. This exemplifies the utility and importance of the ability to incrementally refine the visualisation as it is explored.

The actual queries that were executed were as follows:

```
select (and (seq (windowwithout (seq (bp) (seq (bp) (bp))) 365)
            0 0
            (first (medstart C0([2378]|9[ABCD]).*)))
        (seq (beginning)
            365 1000000
            (first (medstart C0([2378]|9[ABCD]).*))))
select (diag K7[567])
```

Here, the first part of the and-clause in the first select-query is the window without three measurements within a year (365 days) immediately followed by (“o o”) the first prescription of a hypertension-related drug (the medstart-clause). The second part of the and-clause ensures that the medication found in the first part does not start before the beginning of the history (1000000 days, or almost 2740 years, is used to approximate infinity). The second select picks out the patients actually having a hypertension diagnosis.

OBSERVATIONS

Looking at the five patients that matched the final query, the explanations for starting medication were as follows:

- *Prototype-induced error*: One patient had been given a diuretic drug (which is often used in the treatment of hypertension) for a different reason than hypertension. This is perfectly normal. The patient had been medicated with this drug for quite some time before developing hypertension, but since hypertension was eventually developed, the patient was selected by the above query, in error.

Nevertheless, the general practitioner found one surprising funding rule violation: The drug was prescribed as fully paid by the government¹, even though this should only be done when treating hypertension.

¹Termed a “blue” prescription in the Norwegian system

- *High risk patients*: Two patients had very high blood pressure, a situation in which it is normal to start medication immediately. In addition, one of them had a history of heart problems in the family.
- *Poor documentation*: The record of the fourth patient that was inspected did not contain sufficient information about why the patient had been medicated. This case may represent a deviation from the medication guideline.
- *Prototype-induced error*: One patient actually had enough controls of the blood pressure, but the prototype had not been able to extract them (the measurements were on the form “195-200/100”², to indicate that the blood pressure had been measured several times at the same consultation).

²Not an actual measurement for this patient.

Altogether, only one of the patients may be a violation of the guideline. The results show two different ways the prototype can select patients in error: By failing to extract information from free-text fields (can be improved), and by selecting cases where manual inspection of the record is necessary to decide that the case does not qualify.

4.3.3 Query 2: Prescribing ACE-inhibitors

In the third session, we mainly investigated the group:

“Patients prescribed ACE-inhibitors without any known indications of use of this kind of drug.”

This was considered an interesting case because it is against recommended practice: ACE-inhibitors are expensive compared to other options, and there should be a medical reason for choosing them (such as angina pectoris, myocardial infarction or heart failure [oEHGDGo4]).

The actual queries that were executed are given below:

```
select-not (diag T(89|90))
select      (and (seq (windowwithout (diag K7[457]) 730)
                  0 0
                  (first (medstart C09[AB].*)))
            (seq (beginning)
```

```
730 1000000  
(first (medstart C09[AB].*)))
```

Here, the first statement (`select-not`) eliminates the diabetes patients. The next statement is constructed in the same way as query 1, above: An `and-construct` looking for the condition to find (patients having two years (730 days) without indications for ACE-inhibitors, and then an ACE-inhibitor), and ensuring that this window is within the history.

OBSERVATIONS

Running the above query on the set of patients having hypertension resulted in a collection of 17 histories. On his own initiative, the general practitioner examined the histories as two separate groups: Those having ACE-inhibitor as their only medication, and those that also were receiving treatment with other drugs related to hypertension. He did not feel that it was necessary to use another query to split the groups.

- Patients without other medications:
 - *Missing codification*: One patient had had an infarction, but this was not properly coded in the health record.
 - *Poor documentation*: Another patient had no indications in the record of why the ACE-inhibitor had been prescribed. The physician suspected that the patient could have had a heart failure that was not documented in the record.
 - *Visitor*: One of the patients belonged to another health centre, and the main part of his/her record was stored in that centre's EHR system.
 - *Missing codification*: There was a record where a heart failure had not been coded properly.
 - *Hospitalised*: One patient had been at the hospital and had the ACE-inhibitor prescribed from there. The notes from the hospitals were not available.
- Patients with other medications:
 - *Side effects*: Because of side effects from other drugs, four patients were switched to ACE-inhibitors.
 - *Lack of effect*: Due to the lack of effect from other drugs, three patients were switched to ACE-inhibitors. One of these was also counted in the

- above group of patients with side effects.
- *Poor documentation*: Three patients were prescribed ACE-inhibitors without sufficient documentation. One of these was counted under “Lack of effect” above; in this case there was a lack of effect, but the expected choice of drug would be a beta-blocker. There should have been documentation of why an ACE-inhibitor was preferred.
 - *Hospitalised*: Three patients had been prescribed ACE-inhibitors at the hospital, and the notes from these stays were not available.
 - *Prototype-induced error*: In one case, the prototype had not been able to deduct the cessation date for a medication, so this had not been taken into account by the query algorithms.

4.3.4 Query 3: Complications of hypertension

During the fourth session, we concentrated on the case:

“Patients with diagnoses known as complications of hypertension, where the first such diagnosis occurs after the first hypertension diagnosis.”

In the visualisation of this group, the patients were sorted on the time between their first hypertension diagnosis and the first complication.

The actual queries that were executed are given below:

```
select (seq (first (diag K8[567]))
        (first (diag K(7[489]|8[079]|9[01]))))
sort   (first (diag K8[567]))
        (first (diag K(7[489]|8[079]|9[01]))))
```

Here, the select query extracts all patients that have their first complication (second regular expression) after their first diagnosis of hypertension (diag K8[567]). The sort query sorts the histories on the time between the hypertension diagnosis and the complication.

OBSERVATIONS

The inspection of the result from this query was different from that of the others, and the findings were of a more general character. One very interesting observation done by the general practitioner was that of how the visualisation can be used to study “forwarnings”: When the time from the diagnosis of hypertension to the complication is short (easily accessible because of the sort), is there anything in the near past that should have warned the doctor of the upcoming critical event? The general practitioner stated that he did not know of anybody who has studied this, and he stated that he was inspired to investigate this statistically. Note that this is an insight classified by Saraiya et al [SND05] to be among the most valuable of insights – those that generate a hypothesis (see section 2.5.1).

4.3.5 General observations

During the sessions, the following general observations were made by the user:

- *Patterns*: The visualisation gave the general practitioner a novel view of the frequency of follow-up visits and the time spent controlling hypertension patients before starting medication.
- *Many deviations from recommended practice*: Seeing the collection of patients made the general practitioner interested in investigating guidelines. He discovered that fewer patients than he thought were treated according to the guideline, but that most deviations were due to legitimate reasons. This observation led to the formulation of queries 1 (section 4.3.2) and 2 (section 4.3.3).
- *Overview*: Using the visualisation parameterised for hypertension patients enabled the physician to review the hypertension-related parts of the history very quickly, which could be useful in the treatment situation.
- *Basis for discussion*: The general practitioner suggested that such a visualisation could be basis for a discussion with his colleagues about what would be the best treatment and coding practices.
- *Professional profiling*: If it had been possible to show only patients registered with one doctor in particular (the user), it would give an indication of his professional profile: How his patients usually are treated.
- *Ideas*: From exploring the patient material, the general practitioner got new ideas, and he stated that this visualisation stimulated his curiosity.
- *Differing practices*: When inspecting the patients treated with ACE-inhibitors, we discovered that these drugs were often prescribed by hospitals. The physi-

cian commented that general practitioners often complain that their patients too often get expensive medications prescribed during stays at the hospital.

Looking at the way the general practitioner used the prototype, we made the following observations:

- *Search*: The general practitioner did not seem very interested in searching. This might be because it is difficult to understand the possibilities of this utility, or because the align operation, bringing similar cases together, covers much of the same needs.
- *Queries*: It seems that the concept of a *query* is hard to grasp. This might be related to that a query is used in many different operations, and that the query language is difficult to understand for a non-technical user.
- *Visibility*: In some situations, the doctor preferred to turn off event filtering. This was because it made it easier to know where to place the mouse pointer to view preceding contact notes. It may be an indication that there should be a possibility to show all contacts regardless of highlighting without showing non-highlighted medications.
- *Graphical User Interface (GUI)-related*:
 - The scrollbar is in an unexpected place for a Mac- or Windows-user, which made the physician overlook the fact that there were more patients to be inspected than those shown in the initial view.
 - At one time, the user had to be reminded what histories he had inspected. One way to remedy this would be to include the possibility to “mark” a history in a simple way (clicking on its ID number to make the number appear in a different colour or annotated with an icon, for instance).
 - The health record view scrolls to the nearest contact note *previous* to the point clicked in the history. It might be that this behaviour should be changed to scroll to the nearest point in time, because the user sometimes looked at the wrong note when opening the record view (this was prior to the implementation of the contact note brushing functionality).

4.4 Summary of findings related to goals

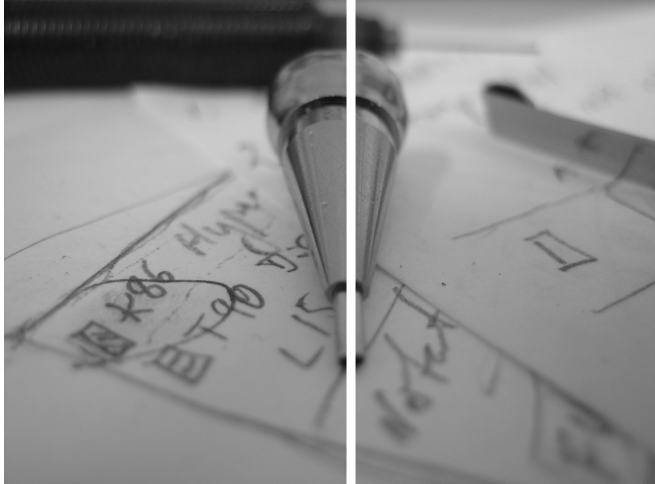
The findings from the case study met our goals as follows:

- *Find possible uses of our design*: The general practitioner found several uses

for this kind of visualisation, as explained in section 4.3.5: The diagram as an overview, as basis for discussion, for professional profiling, for comparison to guidelines, for investigating forewarnings, and for getting ideas about interesting properties of a data set.

- *Investigate the usefulness of the interaction features of the prototype:* Since we were assisting the user in many operations, the foundations for such investigations are diminished. Nevertheless, we were able to realise the importance of easy access to the contact notes, and we discovered interface design problems related to the placement of the scrollbar and the auto-scrolling behaviour of the contact viewer.
- *Look for limitations and problems with, as well as possibilities for improvements of, the prototype:* We improved extraction of blood pressure measurements during the course of the study. There were problems related to usability, and to interval deduction.

At large, we have performed a study that has increased our understanding of the problem at hand and shed light upon some limitations of our current prototype. However, the study is not of a character allowing us to draw conclusions about the validity of our design.



Discussion

(chapter five)

This chapter discusses the results from the case study and positions the proposed design implemented in the prototype in relation to similar work. Additionally, improvements are suggested. The following topics are discussed:

1. *About the informal evaluation:* Discussion of the results from the case study, and indications of the clinical relevance of the proposed design.
2. *Relating the proposed design to other work:* Relation of the proposed design to taxonomies, and to the state of the art.
3. *Suggested improvements:* Concrete suggestions on how the proposed design and the prototype can be improved.

5.1 About the informal evaluation

This section discusses the findings of the case study (chapter 4) and indicates the clinical usefulness of this type of visualisation.

5.1.1 Case study

Concerning the case study, the discussion is split into two parts: A summary of how the prototype was used, and a discussion of the observations made about the data.

TOOL USAGE SUMMARISED

Primarily, it seemed that the general practitioner was using the prototype as a navigational aid to first locate interesting or abnormal situations, and then read the contact notes in order to understand what was going on. This was greatly helped by adding a dynamically changing contact note view to the main window, so that he could inspect the notes without opening the record view. However, this view did not replace the textual record view, as he sometimes still opened the full view – apparently to be able view several notes at once.

It is interesting to note that the user did not think of using the search operation. This might be because he simply forgot: There was a lot of information given in the demonstration. Also, the align operation brought the interesting cases close to each other, highlighting them with a box, rendering the search operation less important. Another reason could be related to the observation we made that the general practitioner had difficulties grasping the concept of a query.

While queries are a common sight in computer science, it could be expected that a physician will have difficulties understanding, much less exploiting, queries in his exploration of the data. Hence, there is a need for an effective metaphor that hides the “queries” and emphasises the operations and medical aspects. We believe this would promote result-oriented thinking and better exploitation of the full potential of the system. It is important to build down technical barriers to free up the user’s capacity for creativity and curiosity: When the user feels in control of the system, to the extent that he forgets that he is using it, his focus shifts from interacting to understanding, and he spends more time analysing and planning than effectuating his plans. On the other hand, textual specification of queries should still be easily

available for the “power user”. Section 5.3.2 discusses various possibilities for user-friendly query specification.

DISCUSSION OF THE OBSERVATIONS

Systematising the findings in chapter 4 revealed eight categories of observations, listed below. Not all categories were relevant for all queries. The categories themselves can be aggregated to compound categories, forming a category hierarchy, as indicated in the following listing:

- *Prototype-induced errors*: Cases where the prototype would extract patients that were not supposed to go into the collection of interest. This could be because of failure to extract blood pressure measurements, or because the time intervals for the prescriptions were missing or not correctly deduced.
- Documentation practices: Categories related to how physicians document.
 - *Poor documentation*: Sometimes, the physician does not properly document why a specific action is taken. This may be just sloppy documentation practice, but it could just as well be the actual cases where the physician did not observe the guideline.
 - *Missing codification*: In some cases, the physician does not enter the diagnosis codes into the correct field.
- Insufficient documentation: Cases where the available information is insufficient.
 - *Hospitalisations*: When patients are treated at a hospital, their contact notes from that stay are not included in the primary doctor’s health record.
 - *Visitors*: Patients that belong to other health centres do not have their full record present and are therefore difficult to analyse.
- Non-deviations: Treatment that differs from the recommended practice for a valid and well-documented reason.
 - Medication-related
 - * *Side effects*: When a drug gives the patient unwanted side effects, an alternative drug may be prescribed.
 - * *Lack of effect*: If the current treatment does not have the desired effect, another option must be tried.
 - *High risk patients*: Some groups of patients need to be treated differently for medical reasons.

We do not believe that deducing categories like the above from the results of our limited study gives results that can act as a valid basis for a statistical analysis. Never-

¹A research project will follow this thesis, further developing and evaluating the design, see section 6.3.

theless is this kind of information important, since we intend to perform an insight-based evaluation in the spirit of Saraiya et al [SND05] later on¹. Before this is performed, a pilot test should be run to discover relevant categories and other parameters for classifying insight; the present classification serves as an example of how this might be approached.

Plotting the few results attained in the case study yields figure 5.1, where the patients examined in queries 1 and 2 are plotted according to their categorisations (some patients are counted twice). The interesting category in this plot is *Documentation practices*, where the cases exhibiting non-compliance to treatment guidelines may be present. In addition, that category may indicate that general practitioners have a potential for improving their documentation practices.

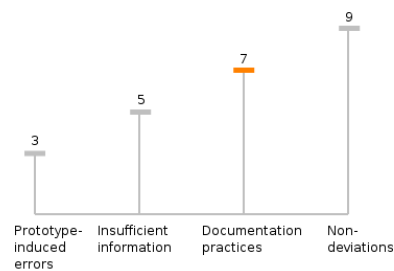


Figure 5.1: Distribution of cases

In subsequent studies, it must be a goal to reduce the impact of *prototype-induced errors* by improving the data processing parts of the program. Regarding the *non-deviations* category, the number of patients being classified in it can be reduced by refining the query, but this must be based on the user's tradeoff between query complexity and the number of histories that are selected. In performing this tradeoff, the user could benefit from the ability to construct the queries incrementally.

5.1.2 Indications of clinical relevance

From the case study, three potential uses within retrospective analysis were discovered:

1. Reviewing practice with respect to guidelines, assessing how often, and why, patients are treated in discord with recommendations.

2. Investigating possible uncaught signals from patients that show up in the office getting one diagnosis, and then face a grave complication only short time thereafter. The general practitioner participating in our study pointed out that there is little research in this area, and that it would be interesting to investigate these situations in greater detail, maybe using statistical methods.
3. Reviewing one's professional profile, i.e how one's patients usually are treated. This can heighten the general practitioner's awareness of his own treatment habits and facilitate comparison of these to guidelines and recommendations.

In addition, it was pointed out during the study that the presentation gave the physician a quick overview of the history of a single patient and an opportunity to review the parts of the history related to a specific problem in very short time. This could be useful in the clinical situation, where the physician could have a better overview of the situation and thus an improved foundation to make the right decisions. In this setting, our visualisation functions as a *LifeLine* [PHL⁺98] – a graphical representation of a life-course history – with contextual information in the form of patients suffering from the same problem.

5.2 Relating the proposed design to other work

To position our design with respect to the literature, we perform an informal assessment with respect to design heuristics and relate it to taxonomies for information visualisation. Furthermore, we discuss how our visualisation resembles and differs from related solutions.

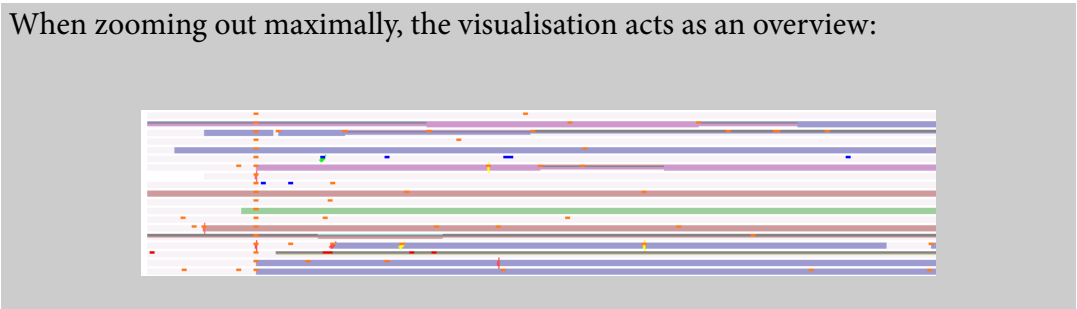
5.2.1 Heuristic evaluation

Referring to the evaluation criteria listed in section 2.5, this section presents an evaluation in terms of recommended practice for interactive visualisation systems. In particular, see figures 2.12 and 2.13 for graphical representations and summaries of these criteria. We carry out the evaluation by describing the prototype in terms of the above-mentioned criteria, to clarify different aspects of the solution, as well as positioning the design in relation to other work in the field.

LIMITATIONS

On a monitor with 1280x1024 resolution and a zoom resulting in each history bar being 20 pixels wide, the prototype is able to display 18 histories simultaneously when no contact note is displayed. Enabling the dynamic journal note display brings this number down to 15. The zoom factor can be decreased, but this makes reading the histories difficult. With a zoom factor enabling one to read histories with up to three medications (11 pixels wide), 45 histories may be read without the contact note. For an overview, a large number of histories can be shown by making each bar only 5 pixels wide. In this mode, the overall distribution of events and medications is shown, but it is difficult to read the detailed medications if a patient has many of them simultaneously.

When zooming out maximally, the visualisation acts as an overview:



COGNITIVE COMPLEXITY

The cognitive complexity is related to the data's density and dimension, and to the relevance of the displayed information. Regarding the latter point, the user is in control of which information is being rendered through the filtering capabilities. Data density is highly data dependent, and dependent on zoom factors. Histories are one-dimensional phenomena, and the dimension of the data is thus unidimensional, though it is multivariate because of the different event types displayed.

SPATIAL ORGANISATION

Data is spatially gathered according to history, and the extents of a history is indicated by a shaded background. Identifying the individual history and which elements belong to which history should therefore be a relatively simple task². Elements may occlude each other when several blood pressure tests are performed on the same date. Events cover medications, but the fact that most medications stretch out for some

²Containedness is a strong cognitive indication of relatedness; see section 2.2.2.

time makes this less of a problem. Horizontally, information is presented chronologically, which is believed to be a logical way to present a history. Vertically, the histories can be ordered using the sort operator to achieve a logically meaningful order. When it comes to focus+context viewing, the prototype lacks support for this, except for an axis showing what time interval is being viewed.

INFORMATION CODING

The information is coded by different symbols for each group of events, and by colouring the symbols differently within each such group. In addition, the icon for blood pressure measurements shows the trend in diastolic blood pressure through the direction of the arrow, and its colour signify whether the blood pressure is considered “normal”, “elevated” or “moderately to severely high.” For the latter category, the intensity of the colour varies according to the value of the test.

STATE TRANSITION

Following a user action, the visualisation is updated in real-time, at least on recent hardware. Since no animation capabilities are present, actions that cause scrolling or re-organisation can be difficult to track. These operations include: Selection (opens a new visualisation with the selected histories), alignment (translates the histories and automatically scrolls to the point of alignment), sorting (the order of the histories is changed, and the chart is (re)aligned) and traversal of search results (the display abruptly changes as new results are scrolled to be within the view). We consider the first three (selection, alignment and sorting) to be operations that are fundamental enough to warrant an abrupt change, since these change the meaning of the diagram. For search, we should consider adding smooth scrolling or at least an indication of when the search has wrapped. An overview (indication of what part of the diagram that is shown) could also be used to aid orientation.

ORIENTATION AND HELP

In the prototype, the user may control the level of detail by using the filtering features, both by adding new types of data to be shown, and by turning on and off display of non-highlighted items. When it comes to undo, there is no explicit support, but the select operation creates a new window (tab) for the visualisation of the selected subset and leaves the original visualisation unchanged. No additional information or help is displayed.

NAVIGATION AND QUERYING

Concerning navigation, simple panning techniques have been implemented through the use of the scroll bars. In addition, there is a zooming feature that controls the width of the history bars and the length of a month. Objects cannot be selected, but object properties are shown when the mouse pointer is positioned over an object, and a detail view of the health record may be opened. Extensive search capabilities are available through the query language, but information not representable as temporal patterns is not available for search. Statistical summary information is not available.

DATA SET REDUCTION

Initially, the data set is pruned through the fact that not all sorts of information present in a patient record database are visualised. Filtering can be dynamically adjusted and is tied to the specification of colours for elements in the visualisation. The data set may also be filtered on the level of histories, by using the select operation. No clustering support is provided.

SUMMARY

By the large, our prototype is best suited as a tool for navigating and inspecting a limited number of histories, but zooming out provides an interesting overview of the general properties of the data being inspected. It could be interesting to investigate how even more histories could be combined with reduced cluttering of the display, by letting the level-of-detail vary dynamically. For example, the diagram could degenerate to a traditional event chart with only key events showing as marks on a thin line. To achieve this, the different events would need to be prioritised.

With event priorities in place, the semantic zoom could be extended to address issues of cognitive complexity as well, mapping it to an interactive slider widget. This could be more valuable than the current on/off toggle for filtering. Focus+context techniques should also be considered.

Concerning information coding, we currently represent a limited subset of the available information. There is still work to do in determining what parts of the remaining information should be visualised, and in deciding on suitable representations for the selected information.

In addition to standard zooming and panning capabilities, the prototype has a query language for describing temporal patterns. This language is used for specification of many of the operations applied to the visualisation. Filtering and selection is possible to give control over the amount of data that is visible, but data cannot be clustered, and statistical summary information is unavailable. The prototype also lacks support for undo and help.

5.2.2 Relation to information visualisation taxonomies

It is useful to position the proposed design with respect to taxonomies, both to show the relation to existing designs, and to gain inspiration from common features of similar systems.

Looking to Shneiderman's Task by Data Type Taxonomy (see section 2.5.3), we need to describe what data types are being represented, and which operations the prototype allows on the various types of data (operations and data types are emphasised in the following text).

The histories are composed of points and intervals in time. In Shneiderman's vocabulary, this is termed *temporal data*. A zoomed-out diagram gives an *overview*. Section 3.3.1 discusses *filtering* and *extraction*, while section 3.3.2 shows how *details-on-demand* functionality is realised. Dynamically adjusting the width of the bars (section 3.2) is a manifestation of *zoom*. Clicking a history in the visualisation to bring up the health record *relates* the two views.

Text is considered to be *1-dimensional data*, and the prototype displays text in the contact note view, and in the journal viewer. The only navigational aid provided is scroll bars. There is a *relation* from the record view to *details* about prescriptions when clicking a medication name.

From the taxonomy described in Müller and Schumann's overview article [MS03], the most significant contribution in this setting would be the characterisation of the time axis (section 2.5.3):

- *Discrete time points vs. interval time*: Both discrete time points (diagnoses, tests) and intervals (medications) are represented in the prototype.
- *Linear time vs. cyclic time*: There is no support for cyclic time, but it could be interesting to investigate how recurring diseases divided into episodes could

be visualised as separate “histories”.

- *Ordinal time vs. continuous time*: The time representation is continuous, i.e. absolute time differences can be computed. This also holds when the histories are aligned, for the individual history, since the time scaling is constant.
- *Ordered time vs. branching time vs. multiple perspectives*: Since the visualisation shows histories from the past, no alternative scenarios are considered.

5.2.3 Relation to state of the art

The proposed visualisation can be seen as an extension of event charts (for example, see [LHD00, AJN⁺03]) with more information shown for each history, not unlike life course visualisation techniques (as in [PHL⁺98, MM00]). Looking at the display of medications, this reminds of the faces of a lexis pencil [FFP, FF96], but differs in the fact that variables (medications) not are assigned to a fixed vertical subdivision (pencil face) of the representation.

In addition, the visualisation is made explorable and designed to be dynamically updated in response to user input. There is also a temporal query language that enables specification of complex patterns for the various data- and visualisation manipulation tasks.

One significant difference from the life course visualisation techniques referred in section 2.4.3 is the density of the information: While the referred techniques use the available screen space for displaying as much information possible about a single history, our design makes a tradeoff between showing relevant information for the individual history and showing a number of histories simultaneously. This implies that our system may require more manual inspection of contact notes to get an overview of a single history, but the brushing feature of the prototype alleviates this. We believe that for retrospective analysis, the prototype may display sufficient information for comparison of the most important aspects of the histories.

It is interesting to compare our design to the work by Fails et al [FKSS06] referred in section 2.4.4, because of the similarity in visual representation³. There are several differences that set the two designs apart:

- Fails et al include what they refer to as “contextual information”, the events that did not match the query whose results are being visualised. This amounts to indication of the number of events at different times. We choose to show the

³[FKSS06] has not yet been published; we came across it at the end of the project, after finishing our implementation and case study. Also, most of this thesis was written at that point.

actual kind of events, enabling more detailed study of relationships between the pattern of interest and its surrounding context.

- Like a traditional event chart, our visualisation shows the entire histories, as opposed to the design of Fails et al showing the time spanned by the hits of a temporal query only. It does not seem clear how their prototype will cope with a situation where the database spans several decades (like ours do), since they do not support alignment of the histories.
- Our timelines can be sorted and aligned, and searches within them may be performed, highlighting the query hits.
- The temporal query language in our prototype is capable of handling parallel and alternative patterns, and the deduction of medication intervals from the individual prescriptions. Also, the window-with and window-without operators are supported to handle time constraints on any query construct. These include features that seem not to be present in the prototype of Fails et al.
- Our diagram is designed to be a dynamic user interface to the data – like a compressed LifeLine. It seems that the visualisation of Fails et al acts more like a graphical rendering of a tabular search result. There is no possibilities for displaying more information than the search results and the number of events surrounding them.
- The patient record is available for the user of our prototype. This was identified as an important feature during our case study, but it is not a feature of the current prototype of Fails et al.
- While we present one patient on each line (focus on the histories), Fails et al show one *match* on each line to avoid overdrawing of overlapping results.

However, there are similarities between the two designs beyond that of graphical resemblance:

- Both designs focus on providing insight into collections of patient histories.
- Temporal query languages play an important role in the exploration of patient data. In both designs, query hits are indicated in the context of the other events in the history.
- Zooming is possible by adjusting the time scale. In our design vertical scaling is also possible.
- Details-on-demand is provided when moving the mouse over an object.

As explained above, the proposed design bears similarities to event charts and life course visualisations. Temporal queries have also been investigated by others. However, we believe that our combination of event chart visualisation of life courses with interactive techniques and temporal queries is novel, and that it is promising as a

helpful technique in the analysis of general practice electronic health records.

5.3 Suggested improvements

We propose several concrete suggestions for improvements that can be implemented. They are presented under the following headings:

1. *Visualisation*: The static aspects of the visualisation, including discussion of perceptual considerations and alternative visual representations.
2. *Interaction*: User interface and query-based operations.
3. *Query language*: Topics related to the query language.

5.3.1 Visualisation

This section discusses the visual aspects of the prototype, with respect to background theory presented in chapter 2. Furthermore, suggestions for improvement are presented.

PERCEPTUAL CONSIDERATIONS

Referring to figure 2.3 (“A visual grammar of diagram elements”, page 17), the visualisation can be evaluated with respect to how well it follows the guidelines for intuitive understanding of diagram elements:

- Entities are represented as closed regions, in accord with the grammar.
- Both shape and colour is used to distinguish between different types of objects.
- Since the event representations are contained within the shaded background of the histories, they classify as related in the language of the grammar.
- In addition to containment within the history, the shapes are ordered on horizontal lines, further strengthening the impression of relation. This also signifies that there is a sequence of elements. The history bar is divided horizontally into two areas; for diagnoses and tests, respectively.
- Concerning size, this is not used intentionally, but the chosen representation for intervals allocates a larger area to each medication when there are few si-

multaneous medications, and this may wrongfully be perceived as reflecting dosage.

Concerning the colours used in the setup for the case study, there are four categories:

- *Foreground colours*: Bright, saturated colours are used to represent diagnoses and tests. These colours include: Red, blue, orange and magenta.
- *Background colours*: Pale colours of low saturation are assigned to medications. These were chosen using a web-designer's tool [col] for creating variants of a colour with the same saturation. A range of six colours are used.
- *Background of the bar*: A light grey is used to indicate the extents of each history.
- *Background of the diagram*: The diagram is drawn on a white background.

It is probably possible to find a better selection of colours by paying closer attention to the guidelines listed in section 2.2.3. Since this involves doing geometry and linear algebra calculations with the colours, we should consider implementing functionality for colour selection in the prototype, or calculate and define different sets of perceptually compatible colours. A quicker solution would be to apply the colours recommended by Ware [War04], listed in the same section.

Background colours should also be given due attention: The current configuration tends to activate the negative space created by the difference in contrast between the bright white background and the grey bars, making thin rectangles stand out between the histories⁴. This could be alleviated by choosing a less bright colour for the background and finding a good balance of contrast between histories and backgrounds. Also, thin outlines with minimal contrast difference to the colours of the bars could be considered.

⁴This is named “1+1=3 effects” by Edward Tufte [Tuf83].

When it comes to shapes, the following are used:

- Long, horizontal rectangles or bars for the outline of the patient histories.
- Small, vertical rectangles for the diagnoses.
- Arrows for the tests.

It should be possible to identify the diagnoses by preattentive processing, at least if they are large enough, since they differ in both size and orientation from the other rectangular elements of the diagram (see section 2.2.2 on choice of shapes). The bars are also distinct. Concerning the tests, it may be that the compound shapes are too

complex for preattentive vision. This last point must be kept in mind when later extensions of the system adds new information to the chart.

On the other hand, whether preattentive processing is the most important concern is context dependent: If the chart is going to be used in a presentation for people unfamiliar with the visualisation, descriptive icons are likely to make it easier to interpret. Also, form plays a stronger role in this setting, where a harmonious-looking image probably will be preferred over an ugly one (as long as it communicates the same information about the data).

One way to activate the preattentive system when wanting to investigate one kind of events, even for icons, would be to blur or whiten everything else: When blurring and/or fading, the remaining sharp and bright features stand out – the search is no longer for specific shapes, but for sharpness and/or colour intensity.

ALTERNATIVE VISUAL REPRESENTATIONS

Based on the discussion above, it is useful to consider alternative visual representations and compare these to the ones chosen in the present prototype.

VISUAL REPRESENTATIONS FOR TESTS AND TEST RESULTS: When depicting tests, there are three properties that need to be communicated:

1. *Test type*: There are several kinds of tests, and these may be organised in hierarchies, e.g. subdivisions of the various sorts of blood or urine samples. The user should be given control of the mapping of tests to representations. All the available mappings should be easily, preferably preattentively, discernible.
2. *Value(s)*: The values of the different tests can be categorised as scalar values (a single number), multidimensional values (vector), nominal values (that belong to a category), or as a combination of these (multivariate). It is possible for a test to have several types of values for the same measurement through abstractions: For example, blood pressure (two scalars) can be classified as low, normal, elevated or high with respect to medical guidelines. This represents a transformation of a multidimensional attribute to a nominal value.
3. *Trend*: In some cases, it is relevant to represent the trend over successive tests. For example, the blood pressure can have increased, remained stable or decreased since the last measurement.

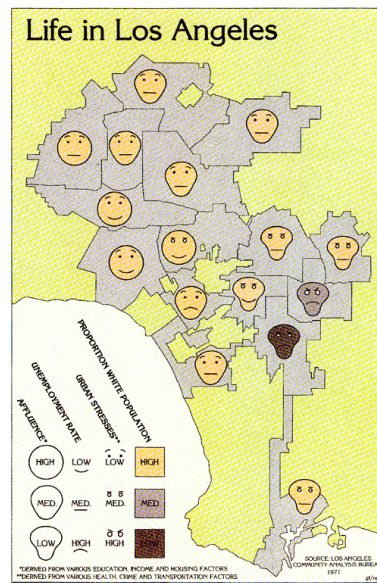
In the case of multi-valued attributes, it may be difficult, and potentially misleading, to depict a change with a single attribute of the symbol: We show the change in blood pressure through the orientation of the symbol (an arrow); the orientation is calculated from changes in diastolic blood pressure only, which means that the (systolic) blood pressure may change without a corresponding change in orientation of the arrow. In our case this leads to confusion in the cases where a change in systolic blood pressure makes the measurement be classified differently than the previous, changing the colour of the arrow (e.g. yellow to red) without a corresponding indication of change of value (an upwards-pointing arrow for increase in blood pressure).

For the display of scalar or nominal values, one possibility is to use glyphs or icons and vary colour, position, orientation and shape according to the test results and trends, or category.

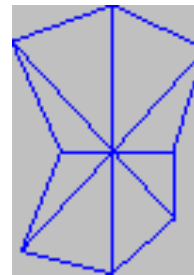
In the case of multidimensional values, object displays may be appropriate. In these displays, each variable is mapped to a feature of the representation: Star glyphs [FGW02] map each variable to a ray emanating from the centre point of a star, and its value decides the length of this ray. The data points are connected by lines to form a star-like figure (figures 5.2(b) and 5.2(c)). Chernoff faces (figure 5.2(a)) use a face with varying shape of head and facial features, such as curvature of the mouth [FGW02].

Ware points out that Chernoff faces may be perceptually unfortunate because humans would tend to interpret the facial features differently and pay more attention to special constellations, such as a happy face [War04]. He suggests choosing object representations where there is a natural or metaphorical relationship between the data and the representation (must be tailored specifically for each type of test). Tufte does, however, warn against misuse of this in his treatment of a graph depicting the relation between launching temperature and damage to space shuttles [Tuf97]. Here, small rockets are used to indicate what may be drawn as a simple bar chart, and it is demonstrated how this clutters the presentation.

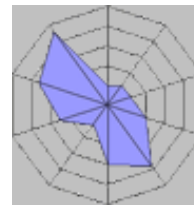
VISUAL REPRESENTATIONS FOR INTERVALS: Intervals should be represented in a way that shows start and end points. The current representation does this. However, when the number of medications changes, so does the width of the bars – without this meaning that the dosage has changed. This could be misleading. Alternatives to the current colouring of background include getting inspiration from *LifeLines* [PHL⁺98] and the visualisations referenced in [MMoo], where interval data is shown using line segments with start and end markers. Another possibility is altering the line style, as



(a) Chernoff faces



(b) Star glyph

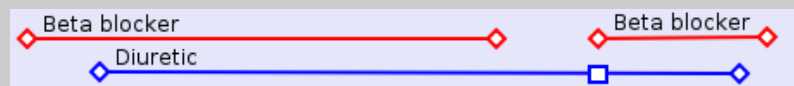


(c) Another star glyph (radar plot)

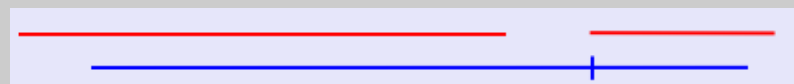
Figure 5.2: Object displays, featuring Chernoff faces (left) and star glyphs (right).

discussed in [LHDoo], or combining the two: Using lines for medications and letting the style of this line reflect the dosage or other parameters of the medication, or to set apart different interval concepts (e.g. sick-leaves, HIV-status).

The following picture shows one alternative representation for intervals:



In this figure, each medication is depicted by a coloured line, with diamonds marking start and end of the medication period and boxes indicating renewal of prescription. The intervals are labelled, but these labels could be omitted to save space, possibly dynamically, based on the available space. Also, the markers can be left out, giving the following representation (the renewal mark is kept, as a vertical line):



This could be seen as an application of the principle of “Occam’s razor”, or an instance of what Tufte terms “The Smallest Effective Difference”:

“Make all visual distinctions as subtle as possible, but still clear and effective.” [Tuf97]

When laying out these lines, one has to decide how they should be distributed vertically. There are several possibilities:

- Let each medication have its own space, and never draw other medications in that position. This is the most space-demanding option, but makes medications easy to identify and compare, as medications can be identified by vertical position as well as by colour.
- Assign the available space to different medications as they begin. This is the simplest option, but the medications will be in a different order at different times. With this choice, there is also the issue of what to do when a new medication starts: Either, the lines for the currently active medications must be broken, or one has to calculate a positioning for the entire history at once. Another option is to count the maximum number of simultaneous medications and space them regularly according to this.
- As a compromise, medications can have a predefined order, and the order shown will always be compatible with the given ordering. However, gives implies the same problems as described above, concerning broken lines.

Another problem is related to cluttering of the diagram, since the bar also must contain diagnoses and tests. This can be solved by widening the bar and reserving an area for medications. If the labels are omitted, it might be an alternative to use pale colours and overdraw the lines.

5.3.2 Interaction

During the development of the prototype, the user interface was not prioritised in areas where the developers could assist the general practitioner in the case study. Before the further evaluations can be undertaken, it needs to be completed so that a non-technical user can use the program unassisted. The need for improvement is most evident in the specification of operations such as select and sort, where there is a command language, and in the entering of queries, which is now performed using

a textual specification. Ideally, the approaches presented here should be verified by low-fidelity prototyping before they are implemented.

ALIGNMENT

The align operation has proven to be a crucial feature of the system: Bringing similar patterns together enables the user to spot differences, similarities and trends. This operation is also used by the select and sort operations.

Presently, the histories are only transposed horizontally, leaving those not matching the specified pattern in their current vertical position. This can result in a diagram of interleaving aligned and un-aligned histories. The usefulness of the operation could be improved if it could bring all histories without the specified pattern out of the way, either by filtering out all histories that do not match, or by moving them to the bottom of the diagram.

SELECTION

With a patient database of 10,515 patient histories, any diagram would become crowded without some form of data set reduction; selection was the first step of all investigations performed in the case study. Concerning the present implementation of the operator, where a new window is opened for each selection, it could be considered to simply delete histories from the current diagram. This could have the advantage of being simpler to understand for the user, but the possibility of quickly comparing a collection with a subset of itself is lost.

SEARCH

Searching was not performed during the case study. The search operation is not easily accessible in the prototype, as a query needs to be typed in. To make the operation available for users, an easy-to-use search specification user interface should be employed. It should be noted, however, that aligning histories serves much of the same needs as a temporal search operator, bringing the search hits close to each other to be inspected using scrolling. Also, the same boxes used to show search hits are used to highlight the actual alignment criterion.

Free-text search could be a useful extension, highlighting all contacts where a given

string occurs in the contact note, and highlighting the search string within the it when it is displayed. In order to realise this, an indexing structure (such as a suffix tree) with appropriate links to the contacts should be employed to enable real-time performance.

A further extension could be to consider indexing the medication names database and transform each free-text search into a search for the corresponding ATC codes (at an appropriate level in the ATC hierarchy). Also, additional information, such as indications for medications, could be included. The same procedure could be applied to all coded information for which natural-language definitions exist.

Another feature of interest could be to highlight search hits and other important items (such as the alignment point) on the scroll-bars, making it easy to know how far to scroll. These indications could also be clickable, scrolling to that point when clicked. If the mouse was held over such a point, there could be a tool tip with a short description of what is highlighted. The horizontal scroll bar could also be extended with small numbers reflecting the content of the axis, so that the user easily can locate any point in time that is of interest.

QUERY SPECIFICATION

Since the visualisation is intended for non-technical users, specification of queries ought to be done otherwise than by entering complex expressions in a query language. One possibility is to have the user construct the queries graphically, using an appropriate metaphor. We see several different possibilities: Building a query tree, specifying the query on a timeline, and specification by example. Textual specification of queries should, however, be available for the “power user”, with good syntax support, encouraging error and feedback messages, and possibly constraints on entering values and help in placing the parentheses right.

An interesting feature for all the following approaches would be the possibility to save a (partial) query to a “scrapbook” and retrieve it later for use in subsequent queries. Another helpful feature would be to dynamically update the view as the search is built (incremental search), so that the user does not need to specify more criteria than necessary, and to speed up the cycle of trying and refining queries.

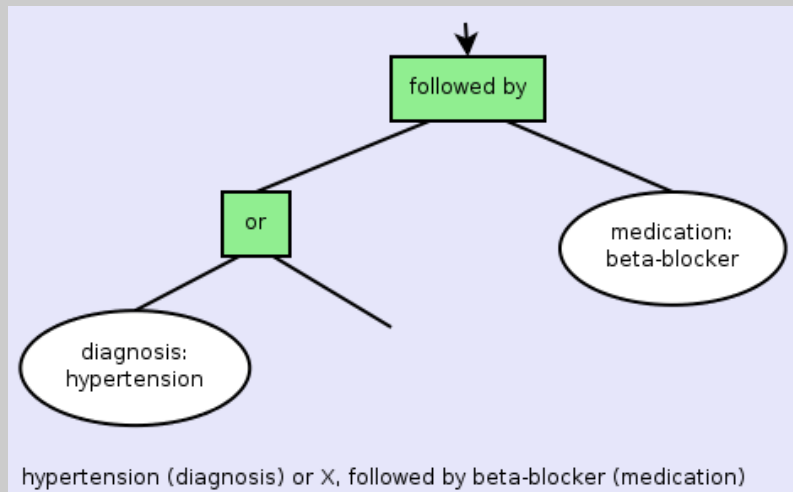
BUILDING A QUERY TREE The user can choose from a toolbox of operations, dragging them to a drawing area. Compound operations are represented as boxes with

sub-queries represented as lines indicating that something needs to be attached. Other compound or primitive operators (represented as ovals) may be attached to such a line, and this is used in building a tree. Parameters to the queries can be specified by selecting a node in the tree and adjusting the values that become available in an accompanying display area.

A natural-language representation of the query can be constructed recursively and displayed. However, when the nesting becomes deep, it can be hard to read such a representation if it is not parenthesised (consider three levels of “or”-constructs ⁵). Alternatively, colour coding of the text could be used to clarify meaning. This could follow the selection in the diagram, so that a sub-query is highlighted when the user clicks a node, or the elements could be coloured according to their level in the query tree.

⁵This could be remedied by letting “or” accept more than two sub-queries.

Building a query tree could look like:



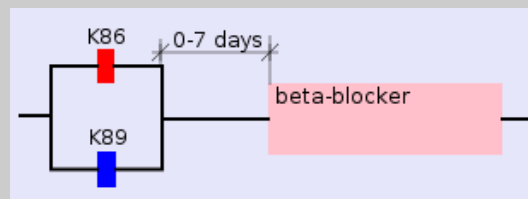
Here, the user has specified a query: (seq (or (diag K86) X) (medinterval C07 . *)), where X signifies an unspecified part of the query. In the interface, seq is displayed as “followed by”, making it easy to read the tree from left to right, and to create a link to the natural-language translation of the tree shown at the bottom.

A critique against this approach is that the notion of trees could not be expected to be well-known amongst users, so that the relation between the tree and the histories may be difficult to grasp.

SPECIFYING QUERIES ON A TIMELINE When the user has learned to interpret the visualisation, one possibility would be to reuse this representation in the specification of a query. The user could click and drag diagnoses, tests and medications to a structural representation of the temporal operators.

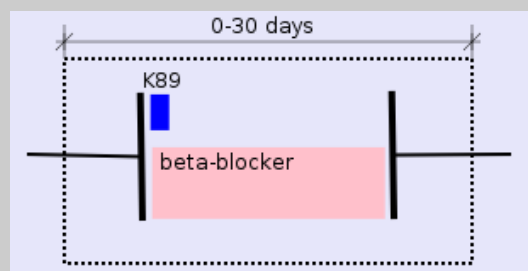
This representation demands that a visual representation language is developed for the different operators. One advantage, in addition to the user's familiarity with the representation, is the flow of time from left to right.

Building a query timeline could look like:



Here, the user has specified the query: `(seq (or (diag K86) (diag K89)) 0 7 (medinterval C07.*))` (the same query as in the query tree above, which lacks the constraint on seq and the K89 diagnosis). The lines that separate and meet again depict the or-operator, and sequence is implied by the left-to-right ordering of the objects. Diagnoses and medications are depicted with symbols similar to the visualisation.

A representation for the operators and and windowwith is shown in the following figure:



Here, the query `(windowwith (and (diag K89) (medinterval C07.*)) 30)` is depicted. The dotted box represents the windowwith, and the two vertical lines signify and (compare to the representation of or, above).

As an abstraction of the representation outlined above, a *petri net* could be used to represent the query. This could produce tidier diagrams for large and complex queries. Another, similar abstraction could be to use a UML activity diagram. However, the similarity to the representations used in the main visualisation is lost with both approaches.

SPECIFICATION BY EXAMPLE Regardless of graphical query representation, the following approach could be used for specification of the queries: When the user creates a selection rectangle in the visualisation (by specifying its two opposite corners with the mouse), a query could be constructed to match the contents of the history at that point, possibly with some fuzziness added by the program. The user could then adjust this query using one of the interfaces mentioned above, to form a basis for searching for or aligning at an interesting pattern that is encountered during inspection of the diagram.

RECORD VIEWING

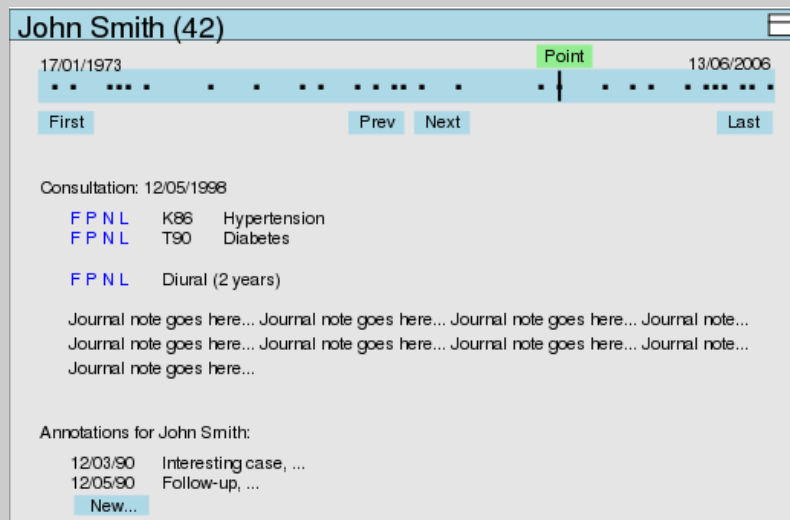
One specific observation that was made during the case study, was that when the general practitioner opened the record view, he did not always read the correct contact note. As an alternative to the present record view, a smaller window popping up *at the point of the mouse cursor* could be employed. This record could be highly interactive, with the following features:

- Diagnoses, medications and tests are annotated with small icons that enable quick jumping to the first, next, previous and last occurrences of that type in the history.
- Clicking a diagnosis brings up the following menu:
 - *Next/Previous*: Two buttons scrolling the record to the next or previous occurrence of that diagnosis, respectively.
 - *Highlight* or *Remove highlighting*: Change colouring preferences for this type of diagnosis.
 - *Align on first occurrence*: Apply an alignment operator to the diagram, aligning all histories on their first occurrence of this diagnosis.
 - *Search for X*: Execute a search for the given diagnosis type (“X” is replaced with the code and name of the diagnosis).
 - *Statistics*: An area of the menu with a different background colour showing key statistics such as prevalence and incidence for the diagnosis in question. Clicking a button in this area generates more detailed statis-

tical reports, such as scatter plots for the distribution of values for the entire collection of patients.

- Clicking a medication brings up a menu analogous to the one for diagnosis codes, with an option to show the full prescription details.
- Clicking a test brings up a menu analogous to the one for diagnosis codes, with an option to do the following: Collect all tests of this type in the current history and draw a curve of values (if the test occurs a minimum number of times and has a real-numbered result value).
- The small record window is navigable with a scrollbar and/or buttons for next and previous contact note. There is also a button to scroll back to the initially viewed contact, which has a shaded background to indicate that it is special (in that it corresponds to the point at which the window is shown).
- Clicking anywhere in the diagram dismisses the record window.
- A special button on the window border turns the mini-record into a separate and persistent window, independent of the visualisation.

The re-designed record window could look like:



Here, the record of the fictitious patient “John Smith”, age 42, is shown. A small timeline shows the extent of the history in years and indicates the contacts. Under this, four buttons for navigation between contacts are shown, and the green button above allows returning to the original contact, at the point of the mouse cursor. It is also possible to click anywhere in the timeline to go to the nearest contact.

Below the timeline, the current contact is listed, with diagnoses and medications. These are annotated with actions for going to the **F**irst, **P**revious, **N**ext or **L**ast occurrence of the diagnosis in question (in an implementation, these could be icons). No tests are shown. The contact note is printed below.

At the bottom of the window, there is room for the user's own comments (not part of the patient record database, see "Annotation", below).

ANNOTATION

During the case study, the physician inspected the histories one by one when he came across something that caught his attention. One way to help the user concentrate on one history, is to blur or fade the others. This is especially useful when switching views, such as when using the textual record. Also, when a history has been investigated, it seems useful to have the possibility to annotate that history with an icon categorising the patient, or at least marking it as seen.

Another interesting feature is the possibility of adding notes and annotations to histories and history entries. This could be inspired by concepts such as WikiMapia [KS], a combination of Google Maps with Wikipedia where maps and satellite photos of the world can be annotated. In a similar manner, diagrams could be annotated with notes showing up as clickable regions, and these notes could be using a Wiki concept to link the information together.

GENERAL USER INTERFACE IMPROVEMENTS

The user interface could benefit from having more linking of displays, and from the exploitation of the visualisation as a user interface. Ideally, most of the frequent explorative operations should be possible to perform with as few mouse clicks as possible, and they should also be easily reversible. For the latter, it is possible to imitate interfaces such as the image manipulation software *GIMP*[Com], where there is a list of actions that have been performed, and where clicking an entry reverses all actions back to that point.

More generally, the system should be extended to better support forgiveness [LHB03, Nor99]:

- *Conventions* should be observed, so that the elements of the user interface works as expected by users (some confuse this with the notion of *affordances* [Nor99]). An example of violating this is noted in the case study: The vertical scroll bar is placed on the left side of the visualisation, and this is the usual place of such a scroll bar (on windows/mac, at least). Placing the scroll bar in an unexpected place confused the user and failed to inform him about that more histories were available at the bottom of the diagram.
- *Undo* should be supported. Using undo makes users feel more safe, encouraging exploration and a faster pace of work [War04].
- Catastrophic errors should have the *minimum impact*, creating a safety net if anything should go wrong. An example of this could be to have the prototype save its state regularly to a temporary file, so that the amount of work lost in a crash is minimised.
- Critical, irreversible operations should be preceded with a *confirmation*. However, this must be used sparingly, as these dialogs can be annoying to the advanced user and introduces an extra step to perform the confirmed operation.
- A *help file* should be available, and the elements of the user interface should be able to document themselves (through tool tips, for instance).

5.3.3 Query language

When we constructed the queries during the case study, we discovered the utility of being able to refer to a sub-query. Several examples of this is shown in the case study chapter, where medications are required to start at least two years after the patients were registered at the health centre. This is solved by repeating the search for the medication in two sub-queries: One to relate the medication to other events and one to relate the it to the beginning of the history. There is a simple way to alleviate this: Adding functionality for defining a name for a sub-query.

Another concern is the exponential complexity resulting from building query trees from the operators defined through the cartesian product (see section 3.4.2).

One way to evaluate the usefulness of the query language would be to collect specifications of “interesting cases” from domain experts, and see if these could be expressed in the query language. Also, this would give an impression of the complexity, in terms of ease of specification and running time, of real-world queries.

COMPLEXITY OF THE WINDOW-WITHOUT OPERATOR

One operator needs to be addressed specifically: The window-without operator. Its definition implies the generation of a large amount of matches, related to the length *in time* of the underlying history. The running time complexity of the window without operator is given in section 3.4.2 as:

$$T(Q_{\text{wwo}}) = T(Q) + O\left(\left\lceil \frac{s(h[|h| - 1]) - s(h[0]) - w}{\delta} \right\rceil\right)$$

This represents a problem, as the running time is not only dependent on the number of elements returned by the underlying query, but on the time spanned by the history and the time resolution in the system. From the preceding expression, it can be seen that when δ decreases, the running time increases dramatically:

$$\lim_{\delta \rightarrow 0^+} T(Q_{\text{wwo}}) = \infty$$

Having a node in a query tree that can generate many solutions can be a difficulty, especially when combined with operators with multiplicative complexity such as the conjunction operator. Note that this is not a property of the implementation, but of the specification. Also note that for a fixed δ , the complexity is linear in the length of the history.

One way to reduce the time complexity of the window-without operator would be to change its specification: Instead of finding windows with a fixed length, it could find windows with a minimum length. This could, however, cause problems higher up in the query tree, as operators could reject the longer windows when they would accepted a shorter one in the right position. A possibility to remedy this would be to define some kind of fuzziness in the interval representation, stating that a match would have a length of the given minimum up to the actual length of the window found. Operators higher up the tree would then shrink the window in one or both ends as new constraints were added, as long as the minimum size would be observed. Alternatively, both forms of the operator could exist.

An important point to observe is that the histories are short, and that it does not seem practical to operate with granularities smaller than a day. At least this holds in a medical setting. If the complexity does not pose any practical problems (it did not during the case study), it would not be worth investing a lot of effort into alleviating this situation.

PREDICATE LOGIC DEFINITION

An alternative approach to implementation would be to define the query language in predicate logic and implement it in prolog, using a deductive database for information retrieval. This would have the advantage that adding new operations would amount to defining them as prolog predicates. In addition, this would allow more flexibility in the specification of queries: The unification mechanism erases the difference between parameters and return values, allowing specification of the return value for return of parameters that would give that result and vice versa.

A naïve prolog implementation of the query language is given in the box below, along with an example database and some queries with results. Since prolog is quite close to a predicate logic definition (as long no cuts are used and the order of the predicates is not important), we do not list both the logic and its implementation.

```

%%
% Implementation of query language in prolog.
%
% The database is assumed to be on the form:
%   diagnosis(HistoryId, IcpcCode, Time).
%   test(HistoryId, TestType, Time).
%   medication(HistoryId, AtcCode, FromTime, ToTime).
%
% Query results are implemented via the predicate:
%   hits(HistoryId, Query, FromTime, ToTime)
%
% Queries are the predicates that end in a "q"
% (andq, orq, diagq, etc).
%%

% Prolog administrativa
:- discontinuous(hits/4).

%
% Concept mappings (unused, for the time being)
%

% History
history(HistoryId) :- entry(HistoryId, _, _).
% Entries: entry(Name, EvTime)
entry(HistoryId, EvTime) :- event(HistoryId, EvTime, _).
entry(HistoryId, EvTime) :- interval(HistoryId, EvTime, _, _).

```

```

% Events: event(Name, EvTime, Type)
event(HistoryId, EvTime, diagnosis) :-
    diagnosis(HistoryId, _, EvTime).
event(HistoryId, EvTime, test) :-
    test(HistoryId, _, EvTime).
event(HistoryId, EvTime, prescription) :-
    medication(HistoryId, _, EvTime, _).
% Intervals
interval(HistoryId, FromTime, ToTime, medication) :-
    medication(HistoryId, _, FromTime, ToTime).

%
% Primitive operators
%

% Point queries
hits(HistoryId, diagq(IcpcCode), EvTime, EvTime) :-
    diagnosis(HistoryId, IcpcCode, EvTime).
hits(HistoryId, testq(TestType), EvTime, EvTime) :-
    test(HistoryId, TestType, EvTime).
hits(HistoryId, prescriptionq(AtcCode), EvTime, EvTime) :-
    medication(HistoryId, AtcCode, EvTime, _).
% Interval queries
hits(HistoryId, medicationq(AtcCode), FromTime, ToTime) :-
    medication(HistoryId, AtcCode, FromTime, ToTime).

%
% Compound operators
%

% Sequence, not constrained
hits(HistoryId, seqq(Q1, Q2), T1, T2) :-
    hits(HistoryId, Q1, T1, T3),
    hits(HistoryId, Q2, T4, T2),
    (T3 < T4), (T1 < T2).

% Sequence, constrained
hits(HistoryId, seqq(Q1, Q2, C1, C2), T1, T2) :-
    hits(HistoryId, Q1, T1, T3),
    hits(HistoryId, Q2, T4, T2),
    (T3 < T4), (T1 < T2),
    (T4-T3=T5), (T5 >= C1), (T5 =< C2).

```

```

% Or
hits (HistoryId , orq(Q1, _), T1, T2) :-
    hits (HistoryId , Q1, T1, T2).
hits (HistoryId , orq(_, Q2), T1, T2) :-
    hits (HistoryId , Q2, T1, T2).

% And
hits (HistoryId , andq(Q1, Q2), T1, T2) :-
    hits (HistoryId , Q1, T1, _),
    hits (HistoryId , Q2, _, T2).

% WindowWith
hits (HistoryId , wwq(Q, W), T1, T2) :-
    hits (HistoryId , Q, T1, T2),
    (T2-T1=<W).

% WindowWithout
% (assumes time to run from -10000 to 10000
% in 1-increment steps)
hits (HistoryId , wwoq(Q, W), T1, T2) :-
    hits (HistoryId , Q, X, Y),
    between(-10000, 10000, T1), plus(T1,W,T2),
    \+ overlaps(T1, T2, X, Y).

overlaps(T1, T2, X, _) :- between(T1, T2, X).
overlaps(T1, T2, _, Y) :- between(T1, T2, Y).

% First
hits (HistoryId , firstq(Q), T1, T2) :-
    setof([A|B], hits (HistoryId , Q, A, B), R),
    smallest(R, [T1|T2], [1000000|0]), !.

smallest([], A, A).
smallest([[X|Y]|R], S, [A|_]) :-
    X < A -> smallest(R, S, [X|Y]).
smallest([_|R], S, A) :- smallest(R, S, A).

% Merge
hits (HistoryId , mergeq(Q, D), T1, T2) :-
    contiguous(HistoryId , Q, D, T1, T2),
    \+((hits (HistoryId , Q, A1, A2),

```

```

        ((A1<T1, T1-A2=<D) ; (A1-T2=<D, A2>T2))).

contiguous(HistoryId , Q, _, T1, T2) :-
    hits(HistoryId , Q, T1, T2).
contiguous(HistoryId , Q, D, T1, T2) :-
    contiguous(HistoryId , Q, D, T1, A2) ->
    hits(HistoryId , Q, B1, T2), B1>T1, D>=B1-A2.

%
% Database
%

diagnosis(1, a, 2).
diagnosis(1, b, 3).
diagnosis(1, c, 4).
test(1, d, 4).
test(1, e, 5).

medication(1, pills , 1, 3).
medication(1, pills , 3, 4).
medication(1, pills , 5, 7).

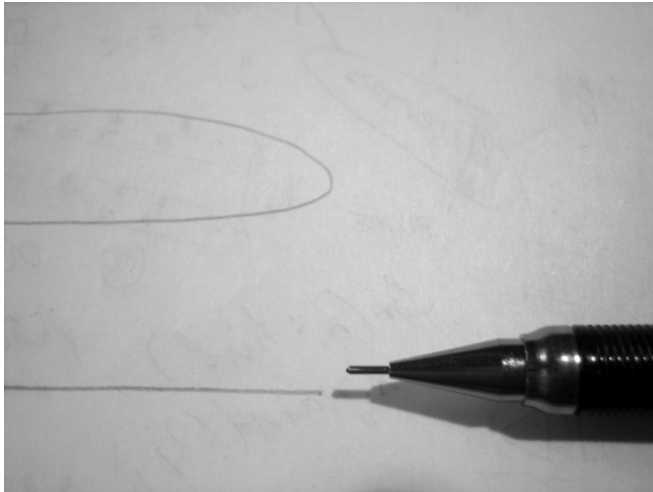
%
% Example queries and results:
%

%% findall([T1|T2],
%%         hits(1, seqq(diagq(E), medicationq(I)), T1, T2),
%%         Ts).
% result: Ts = [[2|4],[2|7],[3|7],[4|7]]
%% findall([A|B],
%%         hits(1, mergeq(medicationq(pills), o), A, B),
%%         Xs)
% result: Xs = [[5|7],[1|4]]
%% hits(1, mergeq(medicationq(pills), 3), A, B).
% result: A = 1 ; B = 7

```

Using the above implementation, we converted our database to a file of prolog facts (on the form of the database in the listing above). ICPC and ATC codes were represented as atoms. We investigated memory requirements. Using GNU prolog [DCoo], the requirements were more than 200 megabytes – at this point, the interpreter crashed.

With SWI prolog [SWI99], we were able to load the database and perform queries on it. The memory requirements were about 43 megabytes for the database, and it took approximately 6 seconds to load. We ran a simple query, extracting all results using the `findall` predicate, and there was a noticeable delay before the results arrived – we measured the CPU time spent to be around 1 second. On the same computer, the query appeared to be executed instantly in the Java implementation.



Conclusion

(chapter six)

This thesis has addressed the following question:

How can we use information visualisation to support retrospective, explorative analysis of collections of patient histories?

In answer to this question, we have put forth a visualisation design and applied it in the general practice domain. The visualisation renders each history as a timeline, indicating events with icons and intervals with the colour of the background. In addition, a set of operations are defined, both on the diagram and the data, supported by direct interaction and a temporal query language.

Our visual representation can be regarded as an extension of event charts with life-course visualisation techniques, such as LifeLines. What we seek can be seen as a middle road between these techniques, incorporating more information into the diagram than a traditional event chart, without going for the complete LifeLine. The extensions applicable for event charts (aligning, sorting, grouping) apply to our design as well.

To investigate the merits of our design, a small case study has been carried out, and its results have led to the formulation of a research project after this thesis is finished. There is still a considerable amount of research and evaluation to be performed before we will have conclusive evidence regarding the proposed design's success in answering the above question.

From the case study, we would like to highlight three findings of the participating general practitioner:

- This kind of visualisation was entirely new to him, enabling him to make comparisons that were not previously possible.
- While investigating the time from diagnosis of hypertension to the first serious complication, the general practitioner remarked that the visualisation inspired him to form a new hypothesis concerning the forewarnings of such grave events.
- The general practitioner stated that this visualisation gave him a new opportunity to review his professional profile.

6.1 Evaluation of our approach

This thesis was approached by early prototyping rather than going through the more common development process of requirements gathering, design, implementation and testing. We chose this path with background in our experiences from previous projects, seeking to realise an idea and collect feedback from users as basis for further improvement. Through this process, we have been able to:

- Develop an idea to a level that enabled us to publish our results¹.
- Decide that the concept is promising enough to be worth the investment in a six-month research project after the work on this thesis has ended (see the next section).
- Obtain valuable feedback from a user, guiding the process ahead.
- Discuss shortcomings of our first attempt, and measures to alleviate these.

¹The paper, submitted to IDAMAP-06 is printed in appendix D.

6.2 Summary of contributions

This thesis has made the following contributions:

- We have proposed a novel design for visual exploration of collections of histories, motivated in a specific problem within general practice health care and existing work in the field of information visualisation. This includes both presentation and interactive navigation of the data.
- We have described a query language and associated algorithms for specifying temporal patterns in a patient history.
- We have developed an interactive prototype to demonstrate our design, and performed a preliminary case study. This case study is not rigorous enough to conclude about the feasibility of the design, but it forms a foundation for improvements of the prototype and further evaluation at a later stage.

6.3 Future work

When it comes to describing the future work, this is treated in two different places: Concrete suggestions for improvement are given in the discussion chapter (section 5.3), while general directions for future research are discussed below.

A research project has been scheduled to develop the design described in this thesis further. This section lists possible directions for that research, whose goals include to perform a more thorough evaluation based on recording of the insights gained through the use of the visualisation, in the spirit of the insight-based evaluation of Saraiya et al [SND05]. The preliminary research plan outlines a process with three iterations, where the prototype is improved based on the experiences from:

- This thesis.
- A small-scale pilot test, to reveal relevant measures for insight in cooperation with a domain expert (see section 5.1.1).
- The final, more complete, evaluation.

In order to fulfil its goal, the future research must address the following issues:

- Investigate relevant clinical problems to be addressed by the design. An e-mail invitation has already been sent to a mailing list for general practitioners, inviting them to submit descriptions of relevant groups of patients that they would like to investigate. Another way to gather requirements would be to arrange a workshop with general practitioners, epidemiologists and computer scientists. The results of this requirements-gathering process would provide a basis for design and evaluation of both the visualisation and the query language.
- Update existing structures (i.e. query language) to enable a stronger relation to established theory. This is important to make the foundations for a publication of the research results more solid. One candidate for a theoretical foundation is the event calculus [KS86].
- Extend and improve the prototype iteratively: Enhance clinical usefulness based on the experiences already gained, and the results from the tests that are performed during the course of the research. Among others, the following topics could be addressed:
 - Improve the prototype's graphical user interface to a level where a general practitioner can use the software without assistance from a computer scientist. In particular, the specification of queries must be addressed.
 - Improve the visualisation and interaction with respect to cognitive issues.
 - Add new features to the visualisation.
 - Improve the interaction scheme, adding new operations and better navigation support, such as focus+context techniques. Also, linking of displays should be considered, and it should be a goal to make use of the diagram as an extension of the user interface.
 - Add complementary analysis techniques, such as statistics and report generation. This could include traditional views of the data such as tables, scatter-plots and curves. Probably, the reports should be generated as results of interactive operations with as little specification of parameters as possible.

Another interesting direction for future research would be to generalise the design and prototype to visualise general histories. The present design does not hinder this. In fact, the prototype handles the information on an abstract level (i.e. histories containing events and intervals) most of the time. However, it should be observed that at one point, specialisation will be needed to address the unique aspect of the different problems. For example, a display tailored for the general practice electronic health record is likely to be easier to read than a display dumping whatever information is present in the history entries that are inspected on a general basis. In practice, this could be approached by using plug-in functionality.

Assignment text

(appendix A)

Visualising collections of patient histories from general practise

In the Norwegian health-care system, the general practitioner plays the role of a gate-keeper to more specialised health services. Also, the list patient system makes patient-practitioner relationships relatively stable. Because of these two factors, Norwegian general practitioner's databases often contain patient histories that are long and relatively complete. There are several, somewhat complementary, ways that knowledge can be extracted from these databases. 1) By reading the journal directly. 2) By statistical analysis. 3) By Knowledge-Discovery and Data Mining (KDD). In this project, we are interested in a fourth way: By visualising information from the database in a way that enables the General Practitioner (GP) (or other user) to use his or her own visual processing system to get an overview and discover features in a collection of patient histories.

In this project, the candidate should design, implement and evaluate a prototype visualisation system for at least one type of patient histories, for example by doing the following (not necessarily in this order):

Select one or more medical problems, for example hypertension or hypotheriosis. For each problem:

- Choose a visualisation technique that will be suitable for the problem.
- Extract, filter and represent data for the problem.
- Implement a visualisation prototype for the problem.
- Empirically evaluate the usefulness of the chosen visualisation technique for this particular problem.

Appendix A. Assignment text

The candidate should consult both relevant literature and clinical experts (see other advisors, below) while undertaking the project.

- Supervisor: Øystein Nytrø
- Main advisor: Ole Edsberg
- Other advisors: General practitioners Anders Grimsmo and Tom Christensen

Abbreviations and terms

(appendix B)

<i>API</i>	Application programming interface
<i>ATC</i>	Anatomical Therapeutic Chemical Classification System
<i>BNF</i>	Backus-Naur Form
<i>CIE</i>	Commission Internationale de l'Eclairage
<i>EHR</i>	Electronic Health Record
<i>ER</i>	Entity-Relationship
<i>GP</i>	General Practitioner
<i>GUI</i>	Graphical User Interface
<i>ICPC</i>	International Classification of Primary Care
<i>IDI</i>	Institute of Computer and Information Sciences
<i>KDD</i>	Knowledge-Discovery and Data Mining
<i>NEL</i>	Norwegian Electronic Handbook for Physicians
<i>NSEP</i>	Norwegian Centre of Electronic Health Records Research
<i>NTNU</i>	Norwegian University of Science and Technology

Complete data model

(appendix C)

Figure C.1 shows the full Entity-Relationship (ER) diagram for the data model of the system. Compare this to figure 3.1, showing the visualised entities only. This figure includes the linking of Entry to Contact and SystemTest.

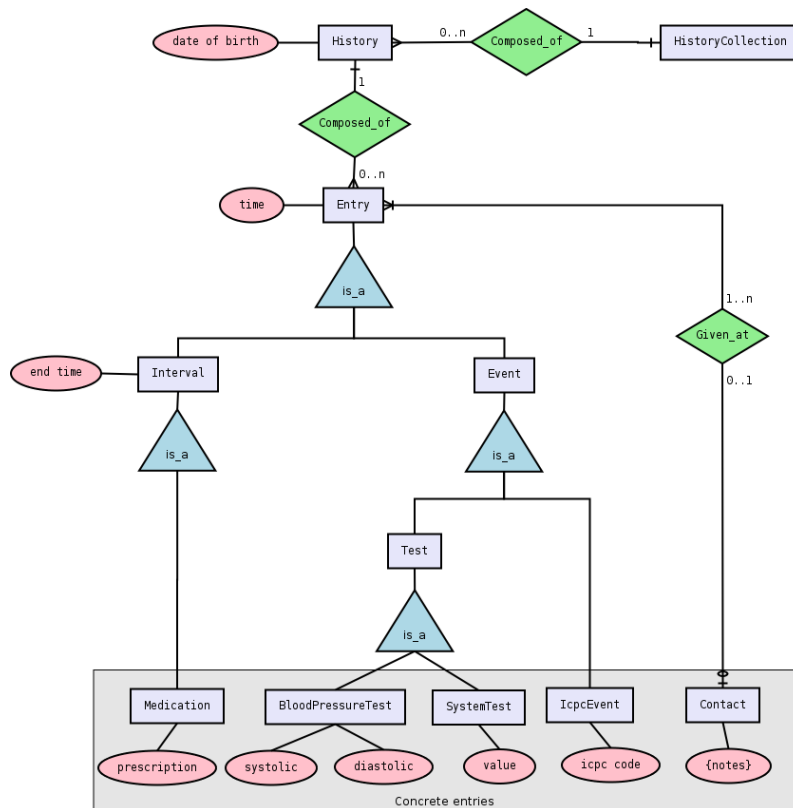


Figure C.1: Entity-Relationship (er) diagram with all entities stored in the system.

Paper

(appendix D)

This appendix contains our paper that has been accepted at IDAMAP-06.

Event Chart Explorer: A Prototype for Visualizing and Querying Collections of Patient Histories

Ole Edsberg^{1),3)*}, Stein Jakob Nordbø^{1),3)}, Øystein Nytrø^{1),3)} and Anders Grimsmo^{2),3)}

¹⁾ Department of Computer and Information Science

²⁾ Department of Community Medicine and General Practice

³⁾ Centre for EHR Research

Norwegian University of Science and Technology, Trondheim, Norway

Abstract

We present our work in progress on a system for visualizing and querying collections of patient histories. The main features of the system are: 1) Compact LifeLines-like visualization of histories as explorable and configurable time lines above a common time axis, with any query hits outlined, and 2) Operations for search, selection, sorting and alignment of the histories based on temporal queries.

1 Introduction

The ability to query and visually explore collections of patient histories is potentially useful in several types of tasks: When faced with a difficult clinical decision, one could search for similar fragments from other histories in the database and explore them to learn from what happened in other cases. In quality assurance of a clinical practice, one could search for deviations from guidelines and explore the result to see if the deviations were justified. In preparing research on clinical processes, one could search for relevant history fragments and explore them to improve one's understanding of the subject matter and get ideas for research hypotheses and analysis methods.

The well-known LifeLines system [Plaisant *et al.*, 1998] provides a time line visualization of the elements of a history. Event charts [Lee *et al.*, 2000] provide a static visualization of a *collection* of histories as a set of stacked and possibly aligned lines above a common time axis, with events represented by glyphs on the lines. The visualization part of our approach can be seen as an attempt to combine the information rich, interactive LifeLines visualization with the event charts' ability to visualize many histories. Related to the query part of our system, the literature describes a query system based on the event calculus that allows users to query collections of series of measurements for patterns of temporal abstractions [Combi and Chittaro, 1999]. Our query language is mainly intended for searching for patterns in the categorical event and interval data of the patient record, and it consequently contains a different set of constructs. We also have a different approach to result visualization.

2 The visualization and query system

Our data model contains point events for contacts, diagnoses and lab results, and interval events for prescriptions. (Our data source unfortunately often requires some guesswork in determining end points of prescriptions.) Figure 1 shows and explains the main view of our system, leaving the query language and query-based operations for the rest of this section. Applying a query to a history results in a set of matches, where a match is defined by its starting and ending points in time. The query language consists of primitive constructs matching data elements in themselves and composite constructs specifying temporal constraints on their sub-queries. Recursively, a query may match:

- a point event, such as a diagnosis, lab test or prescription, or
- an interval of medication with a specified drug type, or the beginning or end of such an interval, or
- a sequence of the matches of two sub-queries, with possible constraints on the time that can pass between them, or
- the parallel or alternative occurrences of the matches of two sub-queries, or
- a window of a specified length, within which a sub-query does, or does not, match, or
- a sub-query's first match in the entire history.

Here is an informal description of an example query: *Find all history fragments where the patient first has a one-year time window without at least three positive blood pressure measurements, and then is prescribed blood pressure-related medication for the first time in the history.* (We omit the syntax, since it is currently under revision.)

The prototype provides the following query-based operations: With the `search` operation, the user submits a query, and red boxes are drawn around the matches of the query. The user can cycle through the matches. With the `select` operation, the user submits a query, and a new tab is opened, containing a visualization of only those histories that contained a match of the query. With the `align` operation, the user submits a query, and the histories are synchronized so that the start points of the matches line up vertically. The time axis changes to show the number of time units relative to the alignment point. With the `sort` operation, the user submits a query with a sequence as the

*Contact email: edsberg@idi.ntnu.no

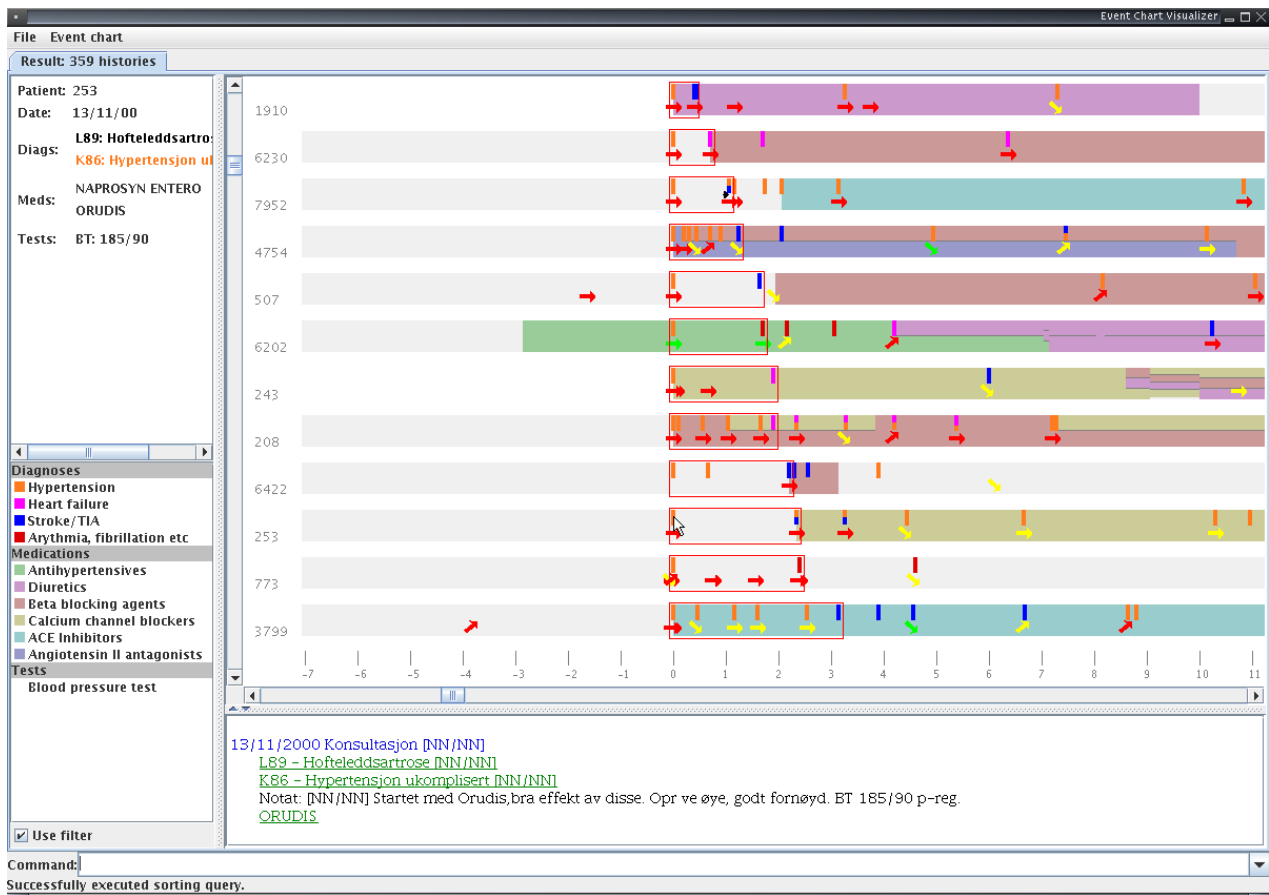


Figure 1: Screenshot of the main view in the prototype. Each of the horizontal bands corresponds to a history. The lower left panel contains a legend of the information types selected to be displayed, in this case specialized for hypertension. In the bands, tall, narrow rectangles indicate diagnoses, coloured subdivision of the background medication, and arrows blood pressure measurements, with colour showing value category and orientation showing trend. The bottom panel shows the journal note and the upper left panel shows details about the events at the current position of the cursor. The search operation has been used to mark the hits of the query informally described as *Find all history fragments where the patient get his first hypertension diagnosis and then, sometime later, gets his first diagnosis for a hypertension-related complication*. The *select* operation has been used to extract the 359 histories containing a hit of this query. Then, the *align* operation has been used to synchronize the histories on the first part of the query. Finally, the *sort* operation has been used to sort the histories according to the distance between first and second part of the query. Menus not shown provide other possibilities, such as zooming, jumping to a journal-like view or changing the information types shown.

top-level construct, and the histories are sorted according to the distance between the matches of the sub-queries. By using these four operations, the visualization can be incrementally narrowed down and adapted to suit the problem at hand. Figure 1 shows a screen shot from our application of the prototype to a data set of about 10000 patient records, in collaboration with a general practitioner wanting to investigate the treatment of hypertension at his health centre.

3 Current work

We are currently working on 1) refining the query language according to our improved understanding of the users' needs, 2) grounding its semantics in the event calculus, and 3) creating a query editor that allows users to design queries in a flowchart-like visual language.

References

- [Combi and Chittaro, 1999] Carlo Combi and Luca Chittaro. Abstraction on clinical data sequences: object-oriented data model and a query language based on the event calculus. *Artificial Intelligence in Medicine*, 17:271–301, 1999.
- [Lee et al., 2000] J. Jack Lee, Kenneth R. Hess, and Joel A. Dubin. Extensions and applications of event charts. *The American Statistician*, 54(1):63–70, 2000.
- [Plaisant et al., 1998] Catherine Plaisant, Richard Mushlin, Aaron Snyder, Jia Li, Dan Heller, and Ben Shneiderman. Lifelines: Using visualization to enhance navigation and analysis of patient records. In *Proc AMIA Annual Fall Symp.*, pages 76–80, 1998.

Digital appendix

(appendix E)

The software prototype that was developed as part of this project is provided as a digital appendix to the thesis. As part of this package, a single directory, *epic*, is provided, containing:

- bin* Binary files (Java class files).
- doc* Application programming interface (API) documentation.
- etc* Data, configuration files and query scripts.
- lib* External libraries.
- src* Java source code (6254 lines¹ in 119 classes, including inner classes).
- build.xml* Apache Ant build file.
- epic.bat* Microsoft Windows batch script for starting the prototype.
- epic.sh* UNIX shell script for starting the prototype.

¹Generated using David A. Wheeler's 'SLOCcount'.

E.1 Running the prototype

To run the prototype, make sure the version of Java installed is at least 1.5.0. Then execute one of the scripts:

Windows

Double-click the “epic” icon, or execute `epic` at the command prompt after changing to the “epic” directory.

Unix

Run `sh epic.sh` from the command line.

Other

It is possible to start the program by invoking Java from the command line as follows: `java -Xmx512m -cp bin no.nsep.epic.ESCviz` (when the current working directory is the “epic” directory).

When the prototype starts, a collection of hypertension patients is loaded, and a query is automatically executed. This query is identical to “Query 3” described in section 4.3.4. Here, patients are sorted on the time from their hypertension diagnosis to the first occurrence of a complication of hypertension. If the visualisation is scrolled vertically, it should produce the same image as in figure 3.3(a), except for the contact note². When moving the mouse cursor over the visualisation, the dynamic detail views on the side and the bottom of the window should be updated.

²Because of privacy concerns, we cannot distribute free-text parts of the EHR. We have, however, inserted the blood pressure measurements that were extracted into the contact note fields.

E.2 Command language

Some of the actions are controlled by entering textual commands and queries into the text field next to the word “Command”:

align <Query>

Align the diagram on the first match of the given query and highlight the match of the alignment criterion.

search <Query>

Highlight (with red frames) all matches of the given query.

select <Query>

Create a new visualisation tab with the histories containing a match of the given query, and align the histories on the first match.

select-not <Query>

Create a new visualisation tab with the histories *not* matching the given query.

sort <Query> <Query>

Sort the histories on the length of the interval between the first matches of the two queries. If one of the queries does not match, or if the second query has a match before all matches from the first, the history in question is moved to the bottom of the diagram.

highlight <RegEx> <Colour>

Highlight all contacts with diagnoses matching the given regular expression, with the given colour. The colour can be specified with a mnemonic name (one of: red, orange, green, blue, magenta, pink, cyan, yellow) or as a hex triplet (“HTML colour”; e.g. “#44ff44” gives a shade of green).

markids <Query> <Colour>

Highlight patient ID numbers with the given colour, for patients matched by the given query.

restart Clear all colour mappings.

loadconfig <Filename>

Load a pre-defined set of highlight mappings. The mapping loaded on startup of the prototype is “etc/hypconfig.dark”.

open <Filename>

Read a file of patient histories (located in the “etc” directory, named with suffix “.dat”).

fromdb <Database> <Username>

Load patient histories from a PostgreSQL database with the given name, logging in with the given user name (no password can be set, and the database must have the structure of a ProfDoc Vision database).

fromcache <Filename>

Load patient histories from a binary file of serialised Java objects (cache files are located in the “etc” directory, named with suffix “.cache”).

tocache <Filename>

Write the current collection of histories to a binary file with the given name.

show Show the main window (called by the startup script, enables us to run database caching operations without a graphical user interface).

Appendix E. Digital appendix

close Close the currently active visualisation tab.

quit Exit the prototype.

source <Filename>
 Execute all lines of a given text file.

The status line in the bottom of the window shows information about the execution of the commands.

Bibliography

- [AJN⁺03] Pamela J. Atherton, Britta Jaspersen, Andrea Nibbe, Kate A. Clement-Brown, Cristine Allmer, Paul Novotny, Charles Erlichman, and Jeff A. Sloan. What happened to all the patients? Event charts for summarizing individual patient data and displaying clinically significant changes in quality of life data. *Drug Information Journal*, 37:11–21, 2003.
- [ASo6] Norsk Helseinformatikk AS. Norsk elektronisk legehåndbok, 2006. Web-based resource: <http://www.legehandboka.no/>.
- [CCo1] Luca Chittaro and Carlo Combi. Representation of temporal intervals and relations: Information visualization aspects and their evaluation. In *TIME*, pages 13–20, 2001.
- [CLRSo1] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, 2001.
- [col] colorsontheweb.com. The color wizard. Web-based service: <http://www.colorsontheweb.com/colorwizard.asp>.
- [Com] Gimp Developer Community. Gnu image manipulation program (gimp). Community website at <http://www.gimp.org/>.
- [Co098] James W. Cooper. *Design Patterns Java Companion*. Addison-Wesley, 1998.

Bibliography

- [DCoo] Daniel Diaz and Philippe Codognet. The gnu prolog system and its implementation. In *SAC '00: Proceedings of the 2000 ACM symposium on Applied computing*, pages 728–732, New York, NY, USA, 2000. ACM Press.
- [FF96] Brian Francis and Mark Fuller. Visualization of event histories. *Journal of the Royal Statistical Society*, 159(2):301–308, 1996.
- [FFP] Brian Francis, Mark Fuller, and John Pritchard. Visualisation of historical events using lexis pencils. Available at <http://www.cas.lancs.ac.uk/alcd/visual/>.
- [FGW02] Usama M. Fayyad, Georges G. Grinstein, and Andreas Wierse. *Information Visualization in Data Mining and Knowledge Discovery*. Morgan Kaufmann, 2002.
- [FKSSo6] Jerry Alan Fails, Amy Karlson, Layla Shahamat, and Ben Shneiderman. A visual interface for multivariate temporal data: Finding patterns of events across multiple histories. In *IEEE Symposium on Visual Analytics Science and Technology (VAST)*, 2006. Not yet published.
- [FLC⁺02] Carla M. D. S. Freitas, Paulo R. G. Luzzardi, Ricardo A. Cava, Marco A. A. Winckler, Marcelo S. Pimenta, and Luciana Nedel. Evaluating usability of information visualization techniques. In *5th Symposium on Human Factors in Computer Systems (IHC)*, 2002.
- [GHS99] Joseph L. Gabbard, Deborah Hix, and J. Edward Swan II. User-centered design and evaluation of virtual environments. *IEEE Computer Graphics and Applications*, 19(6):51–59, November/December 1999.
- [Gri] Anders Grimsmo. Aggregering av icpc-1 koder til større diagnosegrupper. Document received on email August 22nd 2005.
- [Hea96] Christopher G. Healey. Choosing effective colours for data visualization. In *Proceedings IEEE Visualization '96*, pages 263–270, 1996.
- [Hea99] Christopher G. Healey. Fundamental issues of visual perception for effective image generation. In *SIGGRAPH 99 Course 6*, 1999.

- [HMM00] Herman, G. Melançon, and M. S. Marshall. Graph visualization and navigation in information visualization: A survey. In Hans Hagen, editor, *IEEE Transactions on Visualization and Computer Graphics*, volume 6 (1), pages 24–43. IEEE Computer Society, 2000.
- [H0001] Helse-og omsorgsdepartementet. Lov om helsepersonell m.v. (helsepersonelloven), 2001. Available at <http://www.lovdatab.no/all/hl-19990702-064.html>.
- [H0002] Helse-og omsorgsdepartementet. Lov om helseregistre og behandling av helseopplysninger (helseregisterloven), 2002. Available at <http://www.lovdatab.no/all/hl-20010518-024.html>.
- [Jop01] Justis-og politidepartementet. Lov om behandling av personopplysninger (personopplysningsloven), 2001. Available at <http://www.lovdatab.no/all/hl-20000414-031.html>.
- [KS] Alexandre Koriakine and Evgeniy Saveliev. Wikimapia. Website: <http://www.wikimapia.org/>.
- [KS86] R. Kowalski and M. Sergot. A logic-based calculus of events. *New Generation Computing*, 4(1):67–95, 1986.
- [LA94] Y. K. Leung and M. D. Apperley. A review and taxonomy of distortion-oriented presentation techniques. *ACM Transactions on Computer-Human Interaction*, 1(2):126–160, 1994.
- [LHB03] William Lidwell, Kritina Holden, and Jill Butler. *Universal Principles of Design – 100 Ways to Enhance Usability, Influence Perception, Increase Appeal, Make Better Design Decisions, and Teach Through Design*. Rockport, 2003.
- [LHD00] J. Jack Lee, Kenneth R. Hess, and Joel A. Dubin. Extensions and applications of event charts. *The American Statistician*, 54(1), February 2000.
- [MM00] Michael D. Maltz and Jacqueline M Mullany. Visualizing lives: New pathways for analyzing life course trajectories. *Journal of Quantitative Criminology*, 16(2):255–281, 2000.

Bibliography

- [MS03] Wolfgang Muller and Heidrun Schumann. Visualization methods for time-dependent data - an overview. In *Proceedings of the 2003 Winter Simulation Conference*, 2003.
- [MS⁺04] Susan B. Martins, Yuval Shahar, et al. Evaluation of knave-ii: a tool for intelligent query and exploration of patient data. In *Medinfo 2004*, 2004.
- [Nor99] Donald A. Norman. Affordance, conventions, and design. *Interactions*, 6(3):38–43, 1999.
- [Nor05] Stein Jakob Nordbø. Visualising collections of event sequences from general practice patient records. Technical report, NTNU, 2005.
- [oEHGDGo4] North of England Hypertension Guideline Development Group. *Essential hypertension: managing adult patients in primary care*. Newcastle upon Tyne (UK): Centre for Health Services Research, University of Newcastle, 2004.
- [PC95] Peter Pirolli and Stuart K. Card. Information foraging in information access environments. In *CHI*, pages 51–58, 1995.
- [PHL⁺98] Catherine Plaisant, Daniel Heller, Jia Li, Ben Shneiderman, Rich Mushlin, and John Karat. Visualizing medical records with lifelines. In *Proceedings of ACM CHI 98 Conference on Human Factors in Computing Systems (Summary)*, volume 2 of *Demonstrations: Interactive Medicine*, pages 28–29, 1998.
- [PMR⁺95] Catherine Plaisant, Brett Milash, Anne Rose, Seth Widoff, and Ben Shneiderman. Lifelines: Visualizing personal histories. Technical report, October 15 1995.
- [SA05] Daniel J. Simons and Michael S. Ambinder. Change blindness. theory and consequences. *Current Directions in Psychological Science*, 14(1):44–48, 2005.
- [Shn96] Ben Shneiderman. The eyes have it: A task by data type taxonomy for information visualizations, September 05 1996.
- [Shn98] Ben Shneiderman. *Designing the User Interface: Strategies for Effec-*

- tive Human-Computer Interaction*. Addison Wesley Longman, third edition, 1998.
- [SND05] Purvi Saraiya, Chris North, and Karen Duca. An insight-based methodology for evaluating bioinformatics visualizations. *IEEE Trans. Vis. Comput. Graph*, 11(4):443–456, 2005.
- [SSP⁺98] Aaron Snyder, Ben Shneiderman, Catherine Plaisant, Dan Heller, Jia Li, Kaiser Permanente Colorado, and Richard Mushlin. Lifelines: Using visualization to enhance navigation and analysis of patient records, November 18 1998.
- [SWI99] SWI-prolog 3.2, February 11 1999.
- [TSK05] Pang-Ning Tan, Michael Steinbach, and Vipin Kumar. *Introduction to Data Mining*. Addison-Wesley, 2005.
- [Tuf83] Edward R. Tufte. *The Visual Display of Quantitative Information*. Graphics Press, Cheshire, Connecticut, 1983.
- [Tuf90] Edward R. Tufte. *Envisioning Information*. Graphics Press, Chechire, Connecticut, 1990.
- [Tuf97] Edward R. Tufte. *Visual Explanations*. Graphics Press, 1997.
- [vRH04] Peter van Roy and Seif Haridi. *Concepts, Techniques, and Models of Computer Programming*. MIT Press, 2004.
- [War04] Colin Ware. *Information Visualization: Perception for Design*. Morgan Kaufmann Publishers, 2004.
- [WHO05] WHO. Atc index with DDDs, 2005. WHO Collaborating Centre for Drugs Statistics Methodology (Norway).
- [WON98] WONCA. *ICPC-2: International Classification of Primary Care*. Oxford University Press, 1998. WONCA International Classification Committee.
- [ØHN05] Øyvind Hauge and Stein Jakob Nordbø. Nsepter implementation

Bibliography

documentation. Technical report, Norwegian Centre of Electronic Patient Records Research, 2005.