Historical Information Science: Is There Such a Thing? New Comments on an Old Idea [1993]

Thaller, Manfred

Manfred Thaller:

Historical Information Science: Is there such a Thing? New Comments on an old Idea [1993]

doi: 10.12759/hsr.suppl.29.2017.260-286

Published in:

Historical Social Research Supplement 29 (2017)

Cite as:

Manfred Thaller. 2017. Historical Information Science: Is there such a Thing? New Comments on an old Idea [1993]. *Historical Social Research* Supplement 29: 260-286. doi: 10.12759/hsr.suppl.29.2017.260-286.

Historical Social Research
Historische Sozialforschung

Other articles published in this Supplement:

Historical Information Science: Is there such a Thing? New Comments on an old Idea [1993]

Manfred Thaller [*]

Abstract: *»Historische Fachinformatik: Gibt es so etwas? Neue Anmerkungen zu einer alten Idee«*. After a summary of earlier arguments for the differences between information handled in contemporary data bases and information in historical information systems, the model previously proposed for that is extended to include the possibilities introduced by digital image processing, notably the processing of digitized manuscripts. From this conceptual model, a blueprint for a data type "extended string" is derived, which proposes to represent historical texts not as arrays, as usual in computer technology, but by a special data type which implies the representation of strings as graphs. Some engineering considerations for the realization of such a graph based handling of strings are given.

Keywords: Source oriented data processing, non linear strings, manuscript processing.

1. An Intuitive Description

1.1 Introduction

During recent years the computer has been increasingly prominent in many of the disciplines of the Humanities. In the majority of cases, however, this meant just that researchers with a Humanities background discovered that they as well could use tools developed for other people.

This author has for a number of years now proposed, that history would ultimately have to go beyond this stage; that history as a discipline would use data which in the very structure of their informational content would deviate from "information" as it is known in the disciplines dealing with phenomena of current society. We will not repeat the arguments[1] for this line of reasoning, which have

[*] Reprint of: Manfred Thaller. 1993. Historical Information Science: Is there such a Thing? New Comments on an old Idea. In *Seminario discipline umanistiche e informatica. Il problema dell' integrazione*, ed. Tito Orlandi, 51-86 (= Contributi del Centro Linceo interdisciplinare 'Beniamino Segre' 87). Rome.

[1] Most recently: Wolfgang Levermann: Kontextsensitive Datenverwaltung, St. Katharinen: Scripta Mercaturae 1991 (= Halbgraue Reihe zur Historischen Fachinformatik Band B8). See also among others: Manfred Thaller: "Zur Formalisierbarkeit hermeneutischen Verstehens in der Historie." In: Mentalitäten und Lebensverhältnisse. Beispiele aus der Sozialgeschichte der Neuzeit. Rudolf Vierhaus zum 60. Geburtstag. Göttingen: Vandenhoeck & Ruprecht 1982,

been given in the papers quoted: we would rather more show, that more recent techniques lend additional arguments to it.

To prepare this, we will just as briefly as possible summarize the position presented in earlier contributions. There our argument has been as follows.

Processing of historical sources is different from the processing of present day data by a number of reasons: on the most abstract level, this is the case because historians, when they start their research, do not really "know" with absolute certainty, what their texts mean. Therefore, historical data should be administered in a way, which closely resembles the basic principles of a printed edition as used in the historical disciplines, particularly in the schools of medieval studies. The source itself, we said, could not possibly be wrong: if a name was spelled differently at two occasions this could have been an oversight of the scribe; it could be just as well, however, that this "scribal error" was just the only trace left of the existence of two individuals, separated by a minor difference in the spelling of their names. This being so, we claimed, genuinely "historical" data processing must keep the source as closely to the uncorrected original as possible. Now, six different orthographical representations of one word quite obviously tend to frustrate computer supported analysis; as does the use of currencies of unknown interpretation, complex references to calendar dates and the like. All these problems, however, occur also in printed editions: where in the best ones, from a historians point of view, therefore

439-54; Manfred Thaller: "Ungefähre Exaktheit. Theoretische Grundlagen und praktische Möglichkeiten einer Formulierung historischer Quellen als Produkte 'unscharfer' Systeme." In Neue Ansätze in der Geschichtswissenschaft. In this HSR Supplement 29, 138-159. Ed. H. Nagl-Docekal and F. Wimmer. Wien: VWGÖ 1984 (= Conceptus Studien 1), pp. 77-100; Manfred Thaller: "Can We Afford to Use the Computer, Can We Afford Not to Use it?" In: Informatique et Prosopographie. Ed. H. Millet. Paris: CNRS 1985, pp. 339-51; Manfred Thaller (ed.): Datenbanken und Datenverwaltungssysteme als Werkzeuge historischer Forschung. St. Katharinen: Scripta Mercaturae 1986 (= Historisch-Sozialwissenschaftliche Forschungen 20), available at <http://www.gesis.org/hsr/archiv/buchreihe-historisch-sozialwiss-forschungen-hsf/hsf-20>; Manfred Thaller, A Draft Proposal for the Coding of Machine Readable Sources, in: Historical Social Research 11 (1986) 4, 3-46, doi: 10.12759/hsr.11.1986.4.3-46; Manfred Thaller. "The Daily Life of the Middle Ages, Editions of Sources and Data Processing." Medium Aevum Quotidianum, 10 (1987), 6-29; Manfred Thaller "Secundum Manus. Zur Datenverarbeitung mehrschichtiger Editionen." In Geschichte und ihre Quellen. Festschrift für Friedrich Hausmann zum 70. Geburtstag. Ed. R. Härtel et al. Graz: Akademische Druck- u. Verlagsanstalt 1987, 629-37; Computing. Ed. P. Denley and D. Hopkin. Manchester: Manchester University Press 1987, 147-56; Manfred Thaller: "Vom Beleg zum Begriff. Der Beitrag der Datenverarbeitung zur Lösung von Terminologieproblemen." In: Ut populus ad historiam trahatur. Ed. G. M. Dienes et al. Graz, Leykam 1988, 237-54; Manfred Thaller: "Gibt es eine fachspezifische Datenverarbeitung in den historischen Wissenschaften? Quellenbanktechniken in der Geschichtswissenschaft." In: Geschichtswissenschaft und elektronische Datenverarbeitung. Ed. K. H. Kaufhold and J. Schneider. Wiesbaden: Steiner 1988, 45-83; Manfred Thaller: "Draft Proposal for a Format Exchange Program." In Standardisation et échange des bases de données historiques. Actes de la troisième Table Ronde internationale tenue au L.f.S.H. (C.N.R.S.) Ed. J.-P. Genet. Paris: CNRS 1988, pp. 329-75; Manfred Thaller: Datenbankensystem St. Katharinen: Scripta Mercaturae 1989; Manfred Thaller: "Have Very Large Data Bases Methodological Relevance?" In: Conceptual and Numerical Analysis of Data Ed. O. Opitz. Berlin etc.: Springer 1989; Manfred Thaller: "Geographische Angaben in einer Historischen Datenbank." Eratosthene-Sphragide 2 (1990).

two types of information are carefully kept: the literal transcription of a text and a complex environment of apparatuses and appendices, which contain the interpretations the editor has for the text he presents to the historian using the edition.

This structure, we claimed, would have to be repeated in historical data processing. As a result we presented the basic architecture of the Κλειω software system, which carefully distinguishes between the strings of characters administered, which are whenever possible taken literally from the corpus of source material, and the expert knowledge, necessary to process such data: which are administered by a huge array of dedicated subsystems, applying specific historical knowledge to the data, when they are processed by the computer. So, if two entries in a source, which we have reason to expect to be related to only one individual, refer to this individual once as *Josephus de Mons Friduinus* and another time as *Joe of Montefreidin* both forms will be in the data base; the various representations of historical knowledge operating in the background being responsible for handling the fact that these two forms might actually be just one name. If a document happens to be produced on the *Tuesday after Esto mihi 1513,* that is precisely what enters the data base: that it has been written on February the 8th of that year remains to be computed dynamically, when the software of the DBMS actually accesses such an information.

If we summarize a little bit more abstractly the position we have taken with this approach, we might say in source-oriented data processing, as we have propagated it over the years: *A database contains strings of characters, which are organized for speedy processing. It does not contain assumptions, however, what these strings of characters symbolize. To access such a database, it has to exist within an environment of expert knowledge administered by the machine.*

Graphically, we have usually described this situation as in Fig. 1.

1.2 New Technologies: Image Processing

In the context of the network of projects, which used these software components during recent years, a lot of work has been done to incorporate image processing as a set of additional capabilities into such a software environment. We try to summarize in the following the four major branches, which we considered as relatively independent approaches to what image processing means within historical research. In all these cases, we restrict ourselves to the processing of digital images.[2] The following paragraphs assume, that the software environment which is being used contains “fields“ of a data type “image”, which can be addressed by any query possible within the available query language: so an image can become a candidate for processing because of any combination of values within the other fields it is related to. We assume, that the typical historical research project uses data bases, which handle medium numbers of images: in the pilot projects which are currently being undertaken

[2] Still a good introduction: Rafael C. Gonzalez and Paul Wintz: Digital Image Processing, Addison-Wesley, 2nd ed. 1987.

approximately 20.000 color images (of 2-5 MB each) or 20.000-100.00 manuscript pages (of 0.5-3 MB) are planned to be handled.

Figure 1: Traditional Architecture of Κλειω Data Bases

Query of Historian as User

↓

Data Base Software

↓

Knowledge Environment

Local Orthography
Applicable Currencies
Local Calendar
Local Social Terminology
Lemmatization of Source Language
etc. etc. etc.

↓

Uninterpreted Strings as Data Base

The experiences from the early stages of these projects can be summarized as follows:

Immediate Image Retrieval. This describes the basic methodology we hinted at, that is, the ability of a software system to display as the result of a query not just the description of an image, but the image itself. To do this within a research environment, a number of practical considerations apply:

- A hierarchical storage administration[3] should be mandatory. This means, that each of the images out of the whole set of 20.000 or 100.000 administered ones can be displayed in the form of a small snapshot without noticeable delay. This is due to the fact that such small snapshots are being kept on the fastest medium within a storage hierarchy, while the bulk of the image data resides on slower media in the background, access to them possibly even implying the manual mounting of discrete storage media, like magnetooptical cartridges.
- *Each* detail of an image has to be available for zooming with an arbitrary size of the zooming step. This is methodologically extremely important. If you provide images and the possibility to look at only four or five predefined details, the editor of a CD-ROM based system considered to be the most important, the editor is still the only one who controls what you are allowed to be interested in within an image. This does not change dramatically the way of your approach to the images, as compared to a printed book. Only when the user has the possibility to

[3] In this context it has been implemented for the first time, to the best of our knowledge, within a joint project of IBM Japan and the National Museum of Ethnology at Osaka. See Jung-Kook Hong and Sigeharu Sugita: A Color Image Database for an Ethnology Museum, in: Heinrich Best et al. (Eds.): Computers in the Humanities and Social Sciences. München: K. G. Saur 1991, 53-60.

control, what details shall be zoomed – preferably zooming with gain of information – access to the material is better than in the printed solution.
- Of course the usual cutting, pasting, mirroring and similar tools are also necessary within a historical working process.

Image Enhancement. Within the experimental system about 30 statistical operations for transforming and filtering image data have been implemented.[4] These methods – as well as the usual false color techniques – can be applied to any image within the database and or to any segment thereof. The application of these techniques within historical research has usually one or more of the following goals:
- Improving the readability of portions of manuscripts, which became unreadable, because either the writing itself or the material upon which it has been written has changed color, resulting in a reduction of contrast.
- The processing of documents, parts of which have been damaged by mould, humidity or similar reasons.
- The processing of items (inscriptions, coins, etc.) where letters which were cut into the surface or are higher than it were partially destroyed by damage to the object; under some circumstances such material can be partially restored.

Image Binding. Many authors would describe this concept by hinting at "Hypertext", or some of the other "Hyper" concepts. We would like to avoid this, as a matter of intellectual strictness. In data processing there is the concept of nonlinear representations of text: these make up the bulk of this paper, following below. "Hypertext" is a phrase coined by Theodor Nelson, for a very specific and consistent model of a textual data type defined on the basis of specific tools out of the general realm of nonlinear – or nonsequential – texts.[5] This model has so far never been fully implemented. Application systems like HyperCard[6] are no implementations of Hypertext, but systems to administer subsets of the general concept of nonlinear structures.

We therefore prefer to give a more precise definition of our approach, than just a global – and wrong – reference to Hypertext. "Bound Images" we call the administration of bit mapped data objects, which are non-linearily related themselves, can at the same time also be parts of an arbitrarily complex network of transcribed information, however. This is done in a way, where each portion of the transcription

[4] These techniques are currently realized with the help of an image processing library, Image Assistant, which has not been released by IBM for public availability yet, which has been made available to the project, however, by the IBM research laboratory at Winchester by special agreement. At this moment we are testing, whether the general layout of the system is flexible enough to allow for the speedy substitution of this library by a similar - also not yet released - product by Digital Equipment, known as DECImage.

[5] For reasons which have to do with the very peculiar funding of this project, it is somewhat difficult to give good bibliographic references. Theodor H. Nelson: Literary Machines has since 1981 been published in various versions, usually by the author himself. A good summary, which is also easily available: Theodor H. Nelson: Managing Immense Storage in: Byte 13/1 (1988), 225-238; on the concept see also Janet Fiderio: A Grand Vision, in: Byte 13/10 (1988), 237-244; see particularly 238.

[6] This difference is quite oftenly ignored in the HyperCard literature: a particularly bad example in Carol Kaehler: *HyperCard Power*, Addison-Wesley 1988, 366-367.

or description in text form, is explicitly related to an area within the bit mapped object. For the sake of completeness we add, that such areas in our implementation may overlap.

The intention behind such a design is the ability to describe in textual form – or by an arbitrarily complex structured description in a factual data base – any object within a manuscript or image. For the practicability of such solutions it is of tantamount importance that the general software system used allows arbitrary and variable degrees of complexity. The examples which have been used for most of our tests are based on descriptions of images, which employ up to eleven hierarchical levels and about two thousand conceptual "fields".

The purpose of these techniques is:

- It shall become possible to search specifically for parts of images. Getting as result of a query *find me a good example of an early modern scabbard* the whole of Altdorfer's "Battle of Alexander" is not what the analytical user of an image is looking for.
- To evaluate the relative position of elements within images. This allows for the systematic processing of questions of the composition of images: e.g., by querying for images, which seem to conflict with the currently assumed canonical rules of medieval painters regarding the placement of motives relative to each other on a picture. In the processing of manuscripts, this allows an analysis of the relative placement of blocks of text, which may help in the differentiation between several layers of writing in a source which has been produced by more than one scribe over time.

Pattern Recognition. Within the projects about which we are reporting here, the application of the techniques which were already mentioned in the section on "Image Enhancement" gave encouraging results in a number of areas, both when applied to color images and manuscripts. More specifically:

- It is definitely possible, to design sequences of image processing steps, which can be applied systematically to segments of true color images selected by the human using the system. Such sequences are able to reduce the original image towards a representation which shows clearly defined polygons with very few different grey scales.
- The same holds true for manuscripts.
- The polygons resulting in both applications, can be related towards and "measured" against ideal types of basic forms.
- While this is *definitely not* sufficient to clearly identify forms in unprepared images and/or manuscripts, a systematic comparison of the similarities / dissimilarities between various sets of images and manuscripts, or their development over time, becomes possible.

1.3 Manuscript Processing – the Next Frontier?

When we try to apply the tools we have just described with regard to images systematically to manuscripts, we can and will aim at the following process in "editing" a manuscript. For the sake of clarity, it should be made explicit, however, that so far we have described experiences we have gotten with software developed already: the following description is dedicated to our next plans for development.

Step 1: A set of manuscript sources is scanned with medium resolution. According to earlier experiences with the integration of WORM disks and magneto-optical devices into traditional workstations, we can assume that the administration of 10.000-20.000 pages of manuscript on PCs of the upper or workstations of the lower range will be fairly easily possible within two or three years from now. For a precise definition of the capacities which we can achieve, we do not have sufficient experiences with the speed possible in scanning such sources with a satisfactory quality.

Step 2: The documents which in that way are preserved permanently in photographic quality, are loaded into a data base: this data base contains at the beginning just a series of document identifications (basically archival numbers) and the scanned images of these documents.

Step 3: When working with such sources, the historian first of all transcribes the manuscripts literally. For this purpose the following tools are at his command:

- By a system of graphically displayed reference coordinates, the historian can bind individual transcribed phrases (or important abbreviations, or individual letters significant for the specific scribe) to the transcription of these portions of the text.
- Is a reading difficult, it is possible
  - To improve it by the means of the image enhancement techniques discussed.
  - But also to get a set of comparable portions of the manuscript transcribed so far. Either by asking for the graphical representation of cases where the possible readings have been transcribed at earlier stages of the working process or asking for cases where such transcriptions are within a given degree of similarity to the graphical form to be transcribed now.

Step 4: As soon as a continuous transcription exists, or parallel to its creation, tools exist to insert into the text symbolic markup, specifying e.g. persons mentioned or topographical entities referenced. While in the design plans of our project a more general notion of markup is employed, *specifically emphasizing that there are situations, where the graphical representation of symbols may be important, SGML (Standard Generalized Markup Language)*[7] would be a good example for a fairly general markup language which could be employed for such purposes. While we would like to stress, that we consider the question of how such markup schemes should be constructed for other than printing purposes to be anything but closed – rather more: not even opened up – one should point to the efforts of the *Text Encoding Initiative*[8] to define common rules for the applications of such markup schemes in specific areas of the Humanities.

[7] Charles Goldfarb: The SGML Handbook. Oxford: Oxford University Press 1991; vgl. Lou Burnard: What is SGML and how does it help? in: Daniel Greenstein (Ed.): Modelling Historical Data. St. Katharinen: Scripta Mercaturae 1991 (=Halbgraue Reihe zur historischen Fachinformatik), 65-79.

[8] C. Michael Sperberg-McQueen and Lou Burnard (Edd.): *Guidelines for the Encoding and Interchange of Machine-Readable Text*, Chicago and Oxford, Draft Version 1.0, 1990.

In our context such markup is instrumental in converting the transcribed text into a component of a structured data base, into which each portion of the transcription is supposed to be parsed immediately after markup has been completed.

1.4 How do those things interrelate?

We have started with a description of why we have during recent years argumented for a specific architecture of data base systems in historical research; we have emphasized, that they have to contain a considerable number of tools to apply specific knowledge to transcriptions of text, the precise meaning of which changes during a project. The more recent developments we described very sketchily above fit into such an architecture very well: actually we expand the model just in two ways. On the one hand, we assume, that such a system is now able to refer from a character-coded transcription to a bit mapped representation and back; on the other we assume, that the knowledge administered by the system as a whole now contains additional items of knowledge, e.g., the set of "ideal forms" of the letters peculiar to a specific scribe. Schematically, we therefore get the result displayed as Figure 2.

Figure 2: Enhanced Architecture of Κλειω Data Bases

Query of Historian as User

↓

Data Base Software

↓

Knowledge Environment

Local Orthography
Applicable Currencies
Local Calendar
Local Social Terminology
Lemmatization of Source Language
Catalogue of Forms of Letters
Catalogue of Sigla
etc. etc. etc.

↓

Uninterpreted Strings as Data Base

↓

Bit Mapped Manuscripts as Bound Images

There remains the original aim of this paper. We claimed, that the line of reasoning, which brought this author for a number of years to the assumption already, that the use of computer techniques in the Humanities would need more than just the application of commercial packages, would get further support, if we consider more recent technologies. Here the type of our argument, unfortunately, has to change abruptly: while so far, we more or less tried to describe fairly intuitively, how such technologies may influence our current work, we can prove the need for more systematic developments only, when we try to enter a serious technical argument. The remainder of this paper is therefore dedicated to an attempt to define a concise implementation of a non-linear data type “text“ for a Humanities’ – and specifically – Historian’s application system, which fully supports the closely bound mixture of transcribed text and bound images. What we describe on the following pages, is a fully non-sequential data type; it is not hypertext, however, but an alternative proposal for a data type “text“.

2. Design Proposal for a Nonlinear Data Type

2.1 What makes a text “historical“?

Speaking on the most general level, we consider a text to be “historical“, when it describes a situation, where we do neither know for sure, what the situation has been “in reality“, nor according to which rules it has been converted into a written report about reality. On an intuitive level this is exemplified by cases, where two people with the same graphic representation of their names are mentioned in a set of documents, which possibly could be two cases of the same “real“ individual being caught acting, which, however could also be homographic symbols for two completely different biological entities. At a more sublime level, a change in the color of the ink a given person uses in an official correspondence of the 19th century could be an indication of the original supply of ink having dried up; or of a considerable rise of the author within the bureaucratic ranks. Let us just emphasize for non historians, that the second example is all but artificial: indeed the different colors of comments to drafts for diplomatic documents are in the 19th century quite often the only identifying mark, which diplomatic agent added which opinion.

What these introductory examples should demonstrate, is, that the text – the computer interpretable representation of a written document – forms in historical research an intermediate layer between two other layers of information. On the one extreme we have abstract factual knowledge about the various entities described in a text, which allows the interpretation of it; on the other there are purely graphical characteristics of the written document, which may carry meaning, but need not do so.

That the second problem is a genuine markup problem is probably obvious: if we use a computer to prepare diplomatic drafts of the 19th century for printing, we obviously need a way to describe a portion of the document as being “written with blue pencil“. Which, at the time of the first transcription is exactly what it says, a literal description of a graphic property, though during the process of research it *may* well acquire a more abstract connotation, like `author=M.Simpson`. This could of course be interpreted as such properties being eminently fitted to abstract rules for markup, because at the time of producing the markup we have not yet the faintest idea what the final representation in print, if any, of the specific graphic property is to be. Quite besides that at least I cannot very well see, how it should become possible to propose a finite list of such potentially significant graphical properties, there is a more basic problem. We all the time have now been speaking about graphical properties which *may* represent some meaning. Which is another way of saying, that this graphic property is purely accidental. To bring it to a point:

almost all the examples given in the discussions on standardization during the last few years dealt with how to tag a structure which is clearly understood and where the graphic representation is accidental. Historical work deals with structures in a text which we want to *discover,* where the graphics we see may be all the clues we ever might get.

The real difficulty behind this might be a somewhat imprecise definition of what a “text“ is to begin with. To me it seems, that in almost all the contributions to relevant discussions, a “text“ is seen as either the starting point for research or the result of research: either what you get from a colleague to make your linguistic, stylometric or whatsoever analysis from or what you are going to deliver to the printer and, potentially, at the same time to another colleague. In the understanding of this document a text needs to be a considerably more dynamic kind of thing, the formally treatable representation of the current assumptions of a researcher about what his documents actually contain.

This on the one hand means, that we have to provide facilities to mark up graphical attributes which *may* acquire substantial meaning; on the other there have to be provisions for a link between a text and a set of assumptions about portions of it. To go back to our initial example: when we provide for marking a portion of a text as representing the “name of a person“, we will also have to provide for a link to some background data base, which contains descriptions of “real“ persons, being represented in the text by all kinds of conflicting graphic representations. State of the art data bases in history actually carry this a step further, by providing separate links between the graphical variation of a name to an algorithm, which is supposedly able to filter out the “accidental“ orthographic variation of the name, before it is being linked to factual knowledge about the person this name is a tag for.

So, a historical text is in this document considered to be a representation of assumptions about some historical reality, containing on the one hand descriptions of graphical properties, which may require interpretation, at the other linkages to representations of knowledge, which are connected to a specified portion of the text.

This simple model has, however, to be extended into two directions. A “historical“ text is in our opinion something which has come to us under a consistent set of circumstances: our interpretation of colored annotations can obviously be valid only within a corpus of materials which came into existence within one bureaucratic unit. Similarly the language of a medieval chronicle can be analyzed, at least in the first step, only within one copy of that chronicle, though it may have been transmitted, with minor variations, in a whole family of texts. At the same time, however, the reality described by the process of formulating a political document can very often be understood only, if two parallel sets of comments upon some drafts, by different branches of the bureaucracy, are interpreted synchronously; and the “story“ told by a medieval chronicle can only be analyzed, if it is seen as complete as possible: though sometimes no single text exists, which contains all the parts of it.

By first approximation this means, we need a mechanism, to administer an integrated document as an entity, which consists of several layers of traditions, each consisting of some “text“ – i.e. a collection of words – which can only be interpreted in the context of some assumptions about the rules applicable to it. As, obviously, for some portions of the fictitious “true chronicle of x“ texts will exist which have survived in conflicting versions, this leads directly to the requirement of a text representation which allows a given portion of text to have more than one equally valid form. We have, therefore, also to provide for a mechanism, which allows a dynamic handling of variants, which enables software, to treat one coherent representation of a text on a computer, as if it would just consist of one manuscript as well as if it would consist of the logical sum of two or more manuscripts. The computer representation of a machine readable text should therefore in our opinion not only make it possible to handle variants, but to treat all streams of tradition combined into a "text" as potentially equal.

Finally, we have to define the relationship between a "text" as a running representation of a surviving document and a "text" as converted into a database according to some abstract model. In our opinion these two representations should be seen as very close to each other, allowing the database to inspect the natural language context out of which its entities of attributes have been derived, and on the other hand allowing the user of the machine readable text to jump from one portion of it to another portion which, irrespective of the language used, deals with the same abstract concept. More pragmatically: if you enter into a historical database a query like "When did the monastery of St. X receive more than five solidi from a single tenant?" we want to see at least the unstructured description of the relevant entries in the administrative records, if not the scanned image of the respective page, and when we encounter in our running text a peculiarly verbose eulogy about a given benefactor, we would like to be able to get all other sources related to that benefactor, be they in the same source or not.

We are quite aware, that the requirements we just described can be met only in part by existing software. We think however, that there is small, if any, sense to concentrate completely upon the task of how to code data in such a way, that the can be handled by present day software. As a matter of fact, tagging a text exhaustively and completely seems to create such an additional overhead, that I see spurious chances at best, that any historian could be convinced to enter all needed tags by hand. So what we define in the text representation committee is certainly no markup, which normally will and shall be entered by a historian: to pretend otherwise would in my opinion be nothing but fictitious. The sense of a standardized text representation at least in historical research – and decidedly there – can only be to create a means for the communication between software systems, not between human historians.

As such, however, we need a medium that does take account of things to come and is broad enough to give software designers some reason, why they should invest into implementing components, which support such recommendations as an exchange format. To do so we need some foresight; which is why we start from a non-existing system.

2.2 A "Historical Text Engine"

To allow us to do all the things specified in a coherent computing environment, we would first of all like to sketch how such an environment should behave.

We assume on the following pages, that all texts are treated as "information strings". A running text consists simply of a collection of linearily ordered strings of this type; a data base or knowledge representation consists of texts which are connected in a non-linear way. As every linear structure can be described as a trivial case of a non-linear one, running texts, (factual) data bases, full text bases, knowledge bases and, as we will see, collections of bit-mapped data objects are all to be considered as specific realizations of a general representation of information. To make that possible, we assume further, that all the necessary string handling operations are taken care of by a "text engine" which relies on other software components to be provided with correct "information strings" irrespective of how they are administered. We will see, however, that links to other "information strings" can be part of any of them.

In any implementation of the following concepts, a "text engine" could therefore be only realized in close connection with other dedicated software systems, which take care of administering the relationships between various information strings. These do not form part of the present considerations. As the definition of the various items of information to be handled requires references to them sometimes, we will, however, just shortly define the three most important tools of that type.

In our concept we did stress the similarity between a running text and a structured data base; indeed we will later see, that we are also considering cases, where one collection of information strings can alternatively but synchronously be interpreted as a running text and as a data base. To make that possible, we assume that besides the text engine, which we cover here the following exist.

A *text administrator*. This is a very primitive program, which does not very much more, than performing I/O on strictly sequentially stored collections of information strings. A text processing system in our concept would use such a text administrator to save and load information strings from background media, which then are processed with the help of tools from the text engine. Whenever we use the term "text processing" in the remainder of this paper, we refer to software, which performs typical tasks of current day text processing, including primitive full text retrieval applications, by using the services of both, text administrator and text engine.

A *data base engine*. This is a family of software tools, which are responsible to administer nonlinear collections of information strings. These software components are responsible for the handling of all problems resulting from the adaptation of current retrieval concepts to handle context sensitivity of queries and uncertainty or ambiguity of structural relationships.

A *knowledge engine*. This is a family of software tools, which are responsible for the administration of all such conversion and transformation processes, which are built upon knowledge as are based upon dictionary-like structures or complex sets of rules. As all information is supposed to be evaluated dynamically, these software components in turn use components of the text engine, when the need to handle information strings arises.

These "information strings" which are treated by our assumed text engine in the environment shared with these other major modules, consist of linked lists of "uninterpreted items", which exist in an "interpretative environment". Whenever an information string is handed to the text engine, it is guaranteed, that the later is supplied with a full copy of an interpretative environment.

While more precise definitions of interpretative environment and uninterpreted items will be given shortly, it makes their respective roles probably easier to understand, when we describe them somewhat intuitively first. As a first approximation, we could consider the interpretative environment as a table of mappings of abstract font commands into concrete printer operations. So when we look at the text engine operation "prepare output on a specific printer" the printing of the string starts with all such applicable parameters regarding printing and spacing, as can be derived from the interpretative environment handed over with the string to be printed. After this, the uninterpreted items are inspected and item by item converted into such strings and/or printer commands as represent their output form in the light of the current interpretative environment. While this is a description of a current day printing process, in our opinion it should get further: the "font" of a text not only being relevant, when it is being printed, but also, when a string in font "A" is compared to a string in font "B".

A very important consequence of this separation between uninterpreted items and interpretative environment has however to be clarified already now. As mentioned initially, we deal not immediately with the question of markup. We assume, however, that the internal representation of a historical texts indeed needs some features, which are inherently like a symbolic markup: i.e., some information about how the text shall be processed, which is interpreted only, when the text is being processed. This produces a subtle difficulty, when we are speaking about non-linear structures of text, where individual parts of the text shall be accessible. As we will see further, the interpretation of character i of a text may depend on some information, that is contained between character $i - 100$ and $i - 90$ of that text. So, if we want to interpret the i character correctly, we would have to know, that information relevant to that character occurs before it in the string. Therefore we assume, that a "string of information", as we define it, is always administered so, that it is only accessed at a point, where it can be guaranteed, that all information necessary for its interpretation is available. More formally, we speak of *entrance points* into a collection of strings, where a complete copy of the interpretative environment for the following character is available. All characters between two entrance points can only be correctly interpreted, when the text engine reads and interprets first all parts of the string of information, which are situated between the nearest entrance point and the character in question.

The importance of this concept can scarcely be overestimated. Indeed, the need to provide a sufficiently but not unnecessarily large number of entrance points, is the main reason, why we distinguish so sharply between a strictly sequential and linear text administrator, a strictly non linear data base engine, which, however, can assume, that from each of its items a path to the nearest applicable entrance point is defined, and a knowledge engine, which handles dictionaries of relatively small information strings, each of which has its own entrance point, as they can be accessed completely at random.

2.2.1 Types of Uninterpreted Items

Strings of uninterpreted items are made up of five different classes of items:
- *Basic items.*

- *String qualities.*
- *String links.*
- *String variants.*
- *Embedded structures*.

The role of these classes of constituents are in turn:

2.2.1.1 Basic Items

These items carry the actual information derived from a historical source. In the most trivial case, they consist of simple character codes. *All* such items, however, are considered to have logically the same rank. That is, a small bit map (e.g. for a non-deciphered language like the Indus hieroglyphs or a non-textual symbol, like a water mark in paper) or a plain ASCII character can both form distinct items of a "string" in our sense. This assumes, that the text engine contains tools, which can sort and compare *all* types of basic items.

The following types of basic items are defined:
- *Simple characters.*
- *Character tokens.*
- *Bit mapped tokens.*
- *Pictures.*

2.2.1.1.1 Simple characters

Simple characters are described by a sequence of *n* bytes per character, *n* defaulting to one in most text engines. It is assumed in this paper, that characters which represent letters, have one case only. For reasons which are given further below, it is assumed that case is just another string quality which does not justify a special treatment. Each simple character is represented by a numeric value, which indexes a table that contains a variable amount of information about the character. That information consists of:
- Sorting position of the character within the table.
- 'Binding' of the character. By this property we define its behavior in conjunction with neighboring items to its left and right within the same string.

2.2.1.1.2 Character tokens

A character token is represented by a traditional – henceforth called primitive – string of simple characters; in most real-world application starting with a common escape character. While being represented by a primitive string, they are conceptually, however, just the same as simple characters: the degree of similarity between two text tokens – as, e.g, expressed by a table of sorting values – is therefore completely independent of the string representation of the tokens. As the two primitive characters "a" and "A" may or may not be considered identical, independent of the code values assigned to them, in a historical text the two text tokens "`\chrismon`" and "`\cross`" may or may not be considered to be identical or similar; there is, however, no inherent relationship created by both tokens starting with the primitive string "`\c`".

2.2.1.1.3 Bit mapped tokens

Bit mapped tokens are tokens, for which all is valid, what has been said about the properties of text tokens. Bit mapped tokens do not consist of a sequence of primitive characters, however, but of a sequence of the form: `escape-character-length-bitmap`. A further difference is, that their similarity is defined not by a tabular listing of their relationships, but the decision rules for the comparison of the bitmaps themselves.

2.2.1.1.4 Pictures

Intuitively pictures are obviously the same as bitmaps: indeed, their internal representation is assumed to follow the same rules, as just given in the preceding section. While a bit mapped token is assumed to be an atomic item of information, a picture is assumed to be a possibly structured entity, which may occur as part of a text, will more often be connected to it, however, by the mechanism described in section 2.2.1.3.5 for text links.

2.2.1.2 String qualities

As mentioned initially, each uninterpreted information string exists in an interpretative environment. This is defined by a number of assumptions, which are true for the first information of the string. The information string contains, besides the basic items discussed so far, which carry the "real" information, indications for a change in any of these assumptions. This implies for the text engine, that all of its constituents are guaranteed to start the processing of a string only at well defined starting points, all operations defined on the strings while parsing along them. While this may seem to be a distraction, we would like to emphasize it here, as otherwise the concept of string quality cannot be understood. Every string of information exists in an environment which defines its

- *modes,*
- *style,*
- *color,*
- *size and*
- *view.*

It should be noted here, that these names have been chosen for intuitive plausibility, as have the examples below. The flexibility of the concepts, however, is to be derived from the abstract definitions given.

2.2.1.2.1 String modes

String modes define the absence or the presence of a set of attributes. That is, a given item of information can have an attribute or can miss it. It is not possible, to have a mode in a certain degree. Every string of information inherits from its interpretative environment a set of default modes. If a certain mode is not defined in the environment, it is assumed to be absent.

The most intuitive example of a string mode is the case of a character. As we defined before, that simple characters are assumed to be caseless, it would completely depend on the interpretative environment, whether the string

`this is a string`

would be interpreted by the text engine as upper or lower case. By interpretation we mean in this and all following examples, the behaviour of *all* components: an "upper case" string would be printed as upper case (if possible on the output device) but its being uppercase would also influence comparison operations (see below).

At any point in a string a mode can be activated or deactivated:

$\boxed{\text{Mode: }\texttt{+case}}$ `this is a string`

would always result in an uppercase string, irrespective of the assumptions of the interpretative environment,

$\boxed{\text{Mode: }\texttt{-case}}$ `this is a string`

always in a lowercase one. The interpretation of

$\boxed{\text{Mode: }\texttt{+case}}$ `t` $\boxed{\text{Mode: }\texttt{-case}}$ `his is a string`

is clear.

As each mode, which is not explicitly defined in the interpretative environment, is assumed to be absent, their number is arbitrary and has not to be known by the text engine. Modes which are encountered in an information string, for which the text engine has no explicit instructions are therefore completely ignored. In the case of

$\boxed{\text{Mode: }\texttt{+case}}$ `t` $\boxed{\text{Mode: }\texttt{-case}}$ `his is a` $\boxed{\text{Mode: }\texttt{+german}}$

$\boxed{\text{Mode: }\texttt{+case}}$ `z` $\boxed{\text{Mode: }\texttt{-case}}$ `eichenkette` $\boxed{\text{Mode: }\texttt{-german}}$

the mode "german" would in most search operations be ignored; in full text or data base applications it could, however, be used as a selection criterion irrespective of the structure which defines the relationship between this information string and all others in the currently administered data; in Anglosaxon text processing applications it could be interpreted as underlining.

2.2.1.2.2 String style

While any item in a string of information can at the same time have an arbitrarily large number of modes, it always has precisely one style. Statistically speaking, the style of an item is handled on a strictly nominal level: there are no assumptions about any relationship between two different styles expressed in the internal representation of styles.

The most intuitive example for the style of a text would be font information, as in the example

Style: `german` `zeichenkette` Style: `basic`

Please note, that this is not exactly the same than the previous example: while in the previous one, “z“ could acquire the mode case, without losing the mode german, no part of Zeichenkette could acquire the style gothic without losing the style german.

2.2.1.2.3 String color

The quality of color is similar to that of style, by its values being mutually exclusive. Its intuitive interpretation is probably obvious and the introductory remarks of this paper show a potential application. For a systematic interpretation, however, it is much more important, that this quality is supposed to represent statistically an ordinal level. That is, it is assumed to be represented internally by ordinal numbers, which allow expressions of similarity. (A similarity to the implementation of the enum concept in the ‘C’ programming language is intentional.) The two strings:

Color: `dark blue` `George Smith`

and

Color: `light blue` `George Smith`

would in most interpretative environments therefore be assumed to be closer to each other, than the strings:

Color: `dark blue` `George Smith`

and

Color: `light red` `George Smith`

It would, however, *not* be possible, to express the difference in the degree of similarity between the two pairs of names. (This is a statistical statement and a definition of the concept of *color*, not a statement about artistic and/or biological perception of colors.)

2.2.1.2.4 String size

String size, too, has a pretty obvious application. It is similar to the concept of color, allows additionally, however, to express a difference in the degree of similarity between two strings of information being compared. In the three fragments:

Charter a Size: `20pt` `\chrismon` Size: `10pt` `In nomine individue trinitatis...`
Charter b Size: `30pt` `\chrismon` Size: `10pt` `In nomine individue trinitatis...`
Charter c Size: `40pt` `\chrismon` Size: `20pt` `In nomine individue trinitatis...`

the chrismon in a is more similar to the one in charter b therefore, than to the one in charter c; the proportion between the sizes of chrismon and main body of script, however, is identical between charters a and c, while both are dissimilar to b in precisely the same degree.

2.2.1.2.5 String views

The qualities so far – with the possible exception of color – may be seen as an attempt to define classical typesetting attributes in a sufficiently systematic way to allow their interpretation on an intermediate level between typographical representation and conceptual understanding. We recapitulate: the color of a note in a diplomatic document *may* ultimately acquire some meaningful, abstract interpretation; at the beginning of an editorial process, however, it will be exactly what it looks like: proof that Mr. X used a blue pencil.

The concept of *string view,* on the other hand, has been introduced to handle phenomena, which often occur in manuscripts, have, however, no generally accepted typographical conventions assigned to them.

Typical examples would be portions of a text, which are legible and obviously part of the original manuscript, but which later have been crossed out, additions being added at the same time, or manuscripts, which have been written by a number of scribes, some of which can be identified, while others cannot clearly be distinguished. Obviously all these properties of a manuscript could in principle be covered by the string qualities given so far.

The tools provided so far, did always assume, however, that each of the qualities would exist alone: a set of binary qualities, exactly one nominal quality, exactly one ordinal and exactly one which allows comparisons of degrees of similarity. To generalize this model, we introduce the concept of a *text view,* which is defined as *View:* `Type, Name, n`. As its first argument, it accepts any of the previously identified text qualities, i.e., `mode, style, color` or `size`. It introduces a named text quality, which has the properties discussed so far. So our previous notations could be seen as shorthand for a more general text view notation. The following equivalences would hold:

Mode: `n`	==	View: `mode, default, n`
Style: `n`	==	View: `style, default, n`
Color: `n`	==	View: `color, default, n`
Size: `n`	==	View: `size, default, n`

The difference between these two definitions is, however, more important, when it comes to actually implementing such a model. We assume, that a text engine optimizes the four *default views* with regard to speed of processing of individual information string. This means, that when one of the default views is encountered in an information string, it will be taken care of by an extremely quick operation. When an explicitly named view is encountered, however, the text engine is allowed to reorganize the interpretative environment to allow for it. (All comparison operations have to allow for `size` sensitivity without loss of efficiency; a comparison which has to allow for five independent sensitivities for `views` of type `size`, however, is allowed to be significantly less efficient than a comparison that handles just the default `size view`).

This differentiation – and more so the space it is assigned – may be a reflection about the author's background in actual program development: we consider this differentiation to be extremely important, however, as, on the other hand we assume, that historical texts can be handled correctly only, if the number of `views` allowed is unlimited.

2.2.1.3 String links

While we consider string qualities to be a more systematic description of classical textual properties, string links define the conditions for assembling individual information strings into larger objects, like texts or data bases. Basically we consider it necessary to embed into a string reference points, from which it is possible to branch to other strings. The intuitive example for this would be a footnote.

As we mentioned initially, we consider a text not so much to be something which primarily has to be printed, but as a representation of the current knowledge about some historical phenomenon. All such points, where it shall be possible to branch from a given point of reference within a text to somewhere else, are therefore meaningful only as being connected to specific operations of the assumed text engine.

These operations are:

- *branches,*
- *text references,*
- *data base references,*
- *knowledge references* and
- *bitmap references*.

2.2.1.3.1 Branches

A branch is the most simple string link. It consists of a pair of addresses, connecting an arbitrary point within a string of information with the entrance point into another string. The intuitive example for it is a note in text processing. Branches pointing from an arbitrary point of an information string to the entrance point into another string, we will call *exceptions* and denote with the symbol $\boxed{\text{Name}\rightarrow}$; branches from the entrance point of a string to an arbitrary point of another information string, we will call reference and symbolize by $\boxed{\leftarrow\text{Name}}$

In the case of a footnote, these elements would be used as follows:

footnotes →	`this point is usually not discussed any more...`
← footnotes	`Cf. John Smith;...`

The text engine resolves the arrows in these string links as follows:

- Exceptions are plain pointers to the beginning of another string of information, allowing the interpretative environment to be initialized the standard way.
- References are similar pointers to the arbitrary point from which the exception did branch away. They can, however, only be traversed, if this point in the collection of strings has been reached via a previous reference from the corresponding exception. In such cases the text engine stacks a copy of the state the inter-

pretative environment has been in, when the exception was activated. If the reference is reached by any other navigational operation within the collection of strings in question, it is not possible to follow it to the spot of the exception.

2.2.1.3.2 Text References

Text references allow it to bracket a specific portion of text and logically to assemble all such portions into a specific collection of texts. An intuitive example within text processing would be the creation of registers.

Text references consist of pairs of the form

$\boxed{\text{Name} \rightarrow}$ `some string` $\boxed{\leftarrow \text{Name}}$

which we shall discuss as "forward reference", "reference string" and "backward reference" respectively.

A *forward reference* consists of

- a mark pointing to the end of the reference string,
- a pointer to the next forward reference in the collection of strings with the same *name* and
- a pointer to the nearest entrance point into the information string containing the next forward reference in front of it.

It enables a text engine therefore, to navigate from one text reference immediately to the next; does not remove the necessity, however, to interpret the portion of the information string in front of the respective forward references to bring the interpretative environment into the state it has to be, when the reference string shall be interpreted correctly.

A *backward reference* contains the same information, does so with respect to the preceding text reference in the collection of information strings in question, however.

2.2.1.3.3 Data Base References

Data base references have no precise equivalent in traditional applications. A specific data base pointer $\boxed{\uparrow x}$ is defined by

- the data base which shall be referenced,
- a procedure specifying for the data base engine in question, how the reference shall be converted into an information string.

When control returns to the text engine, a modifiable copy of the information string created in this way is brought to its disposal and for purposes of text processing integrated transparently into the "text" proper. Obvious, but trivial examples of usage would be the dynamic treatment of bibliographical and/or biographical information.

2.2.1.3.4 Knowledge References

Knowledge references are denoted by identical bracketing reference symbols of the form

`↓KB object text ↓KB`

Object text refers to an information – like complex chronologies, historical currencies and similar, as described in previous papers by this author – which needs to be transformed, before submitted to specific classes of treatments. (Think once more of sorting or comparisons.)

The *knowledge* referenced is assumed to consist of a formal definition of the format an object text has to have to be processed plus a collection of suitable dictionaries to apply specific transformation rules.

A specific ↓x consists of

- a specification of the knowledge (base) to be accessed plus
- a set of operations of the text engine, which trigger the transformation of the object text.

For all such operations, the text engine does not process the object text itself, but the result of the conversion.

2.2.1.3.5 Bitmap References

A bitmap reference consists of a pair of identical bracketing symbols of the form

`||x interpretable equivalent ||x`

It consists of

- a reference to a bit image, which either is administered as a "picture" as defined above or as a completely independent file and
- a set of coordinates within the bitmap.

The text engine references the bit image, when suitable display units are available, uses the interpretable equivalent, however, when functions are being called, which require operations like search or comparison.

2.2.1.4 String variants

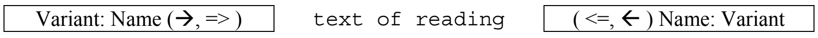
String variants have an obvious application: the administration and processing of the *apparatus criticus* of a text. The following model for the integration of variance into a general text representation model sees this only as a starting point, however. We rather assume as application the creation of dynamic text representations, i.e. of text representations, which allow software to treat all variances of a text as "equal". This could be envisaged by display modules, which allow the user to switch from variant α to variant β by hitting a function key, the displayed text in all cases where variants exist in both stages following the readings of manuscript α or β.

It is assumed that – unless indicated otherwise by the interpretative environment – an information starts with a part that is present in all witnesses for a given text. This assumption is changed, as soon as in the processing of the information string a *block border* is encountered. Block borders, which can be nested, limit parts of an information string, for which substitutable variant readings exist.

Generally a text with variants would therefore be represented as

`Text present in all witnesses Block >> variant reading << Block text present in all witnesses`

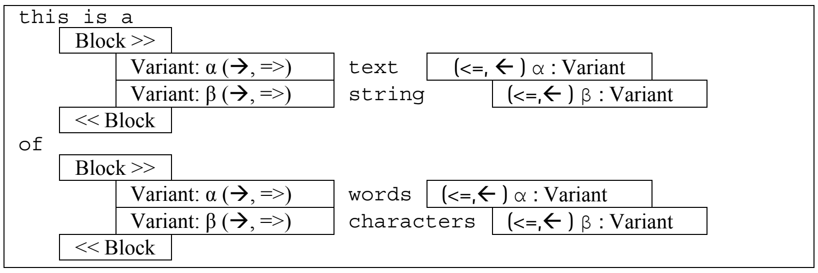
The *variant readings* consist of blocks of the form



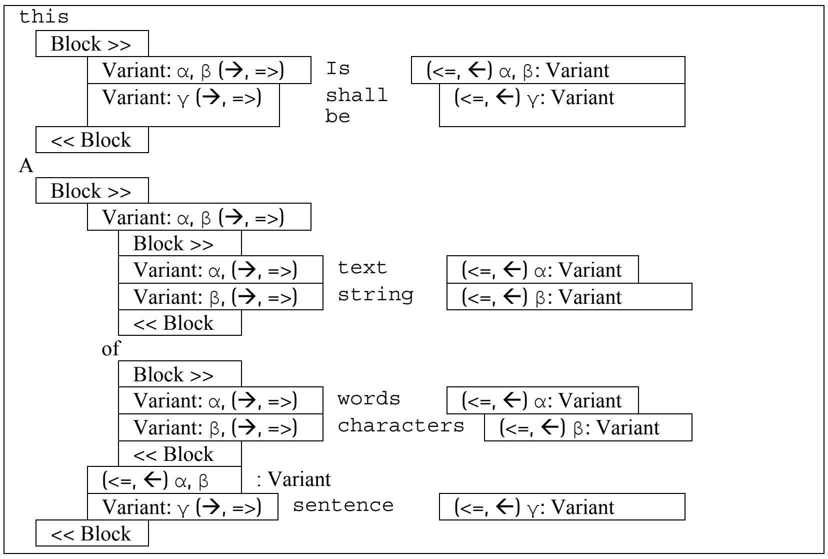
where *nameset* specifies in which witnesses a given reading is present.

The (→, =>) and (<=, ←) symbols indicate, that all variant readings are linked as a list, where each atom has a pointer to the next variant reading, but at the same time also a pointer to the end of the respective block to speed up processing.

The mechanism is probably best explained by two examples. Let's first look at the situation where manuscript α gives `this is a text of words`, while manuscript β reads `this is a string of characters`. In this case we assume a representation which is:



When, to show nesting, we add manuscript γ which reads this shall be a sentence, we get:



As this may look somewhat complicated: let me remind the gentle reader, that we are describing here a structure, which internally shall be able to handle something. Entering markup into a text, which in turn is parsed to provide the required pointers, would be one way to achieve this.

2.2.1.5 Embedded Structures

While the preceding parts of this paper form presumably a general model for the representation of historical texts, this section may be more specific to the activities of the author. It deals with the problems from representation, where a data base, which follows a network model, is integrated into a text. The idea is, that data base queries are processed, which are translated via structural information contained in a data dictionary into such navigational processes as are required to select various items of information for processing. Unlike in traditional data bases, that information is, however, not "extracted" from a data base. Instead of extracted information from natural text, we assume, that the text as a whole is represented on the machine, various pointers – similar to the classes of such, which we described so far – being "hidden" within the running text, which define that a given part of it is not just a series of characters, but at the same time the content of a structurally defined field of a data base.

The idea is, that we have a text, like the sentence *On the 19th of March 1764, John Winslow and George Bilmington appeared before this court to claim* .... which by software based upon our ideal text engine can be handled for typical purposes of text processing, other software based upon this engine can at the same time, however, treat terms contained within it – like *John Winslow* – as the value of the attribute name of an incidence of the entity `person`. Applications would be tasks like: *Select all court records, where people who have been born more than fifty miles from a given city of reference appear as interested party; select out of the running text all portions which are marked as "quotations"; compute a set of stylometric indices for the vocabulary used in such quotations*.

As already mentioned, we do not claim in this section, that our considerations here are general. We assume, that such models of the combination of structured representations of information and the covering text containing that information, can only be realized with data structures that allow inconsistent information to be handled and avoid normalization procedures.

As such models are few, we have so far only considered the problem of "hiding" Κλειω data structures within the text. In the case of that system, the data are structured into a network which consists of entities[9] called *groups* which among themselves can be linked by arbitrarily many relationships. Groups have an arbitrary number of attributes, called *elements,* which conceptually consist of variable length arrays allowing an arbitrary number of values, called *entries,* for each attribute.

To hide a network like that within a collection of information strings, as we described them so far, we see two possibilities:

On the one hand, the structural network of a data base could simply exist parallel to a text as we described it here, all entries of the network consisting of pointers to

[9] We discuss here the problem of hiding data structures within a general text representation. Κλειω groups are not entities in the sense of conventional DBMSs: to clarify, that they are not, we call them groups; as a precise definition of the differences would go beyond the scope of this paper, we do not give it however, but mention that envisaging them as entities will not be totally wrong. The same is the case for "elements" and "entries" introduced below.

the relevant portion of the text, together with length information. (Obviously these pointers would have to consist of the nearest entrance point of the text plus an offset to the relevant portion.) The advantage of such a construction is obvious: existing database software could be taken over more or less unchanged and all textual functions needed, are contained per definition in a text engine, as we defined it so far. In our opinion this otherwise obvious model has, however, two major shortcomings: obviously updating the text could easily corrupt all the addresses contained in the entries of the data base, serious, as this problem is, we could handle it conceptually, by allowing for $\boxed{\uparrow \text{DB}}$ links with a slightly modified definition, which would have to be inserted into the text at each point, which is the target of a data base referencing it. While this update problem is certainly tricky, it could eventually be mastered by the techniques hinted at.

Conceptually much more severe is a problem, which is introduced, when we consider even the most simple concept of data types. We already have introduced in section 2.2.1.3.4 above the concept of a *knowledge pointer* $\boxed{\downarrow\text{KB}}$. We did so, because of the necessity to treat "special kinds of text" in a special way. It is not sensible to sort calendar dates in alphanumeric fashion; and converting temporal notation related to the medieval saints into a notation that can meaningfully be sorted is no completely trivial task. This, however, is a typical data base problem. Of course a number of solutions could exist: one would be to include the functional equivalence of a $\boxed{\downarrow\text{KB}}$ into the data dictionary for this field – which means a type of redundancy which is dangerous. Another solution would be, to let the $\boxed{\downarrow\text{KB}}$ point to the data dictionary, rather than to the relevant knowledge bases to begin with. This could mean, that we need different implementations for a $\boxed{\downarrow\text{KB}}$ that occurs in a text without a hidden structured data base than the one used in texts with such an underlying structure; a bad solution. Avoiding that, however would bring us into the situation, where texts without an underlying dummy structure would not be allowed; not a bad solution, but an outright impossible one.

To avoid these situations, we propose therefore to discuss data models, which are per definition built into "natural texts". This would seem unusual from the point of view of other disciplines, but in history – or rather in the source-oriented model of historical data processing – data bases, as this author has pointed out elsewhere, are always not so much collections of information, which define their own reality, but an attempt to structure information that has survived in a coherent form, i.e., usually as a text.

In the case of Κλειω we consider this possible, by allowing the text engine to administer three additional classes of constituents of a string of information: *group pointers, element pointers* and *entry pointers* (denoted as $\boxed{\text{Group:}\texttt{Name}\ \bullet}$, $\boxed{\text{Element:}\texttt{Name}\ \bullet}$ and $\boxed{\text{Entry:}\texttt{Name}\ \infty}$, respectively). They are defined as a generalization of the concept of a text reference introduced in section 2.2.1.3.2. The ∞ symbol replacing the =>/<= component of the $\boxed{\text{Name =>}}$ / $\boxed{\text{<= Name}}$ notion is assumed to indicate that this generalization allows for an unlimited number of references starting from each embedded structural symbol, while our textual references provided always for precisely one physical reference.

2.2.2 The Interpretative Environment

The interpretative environment, which we have known intuitively so far as a kind of refinement to the concept of a printer driver consists of two independent components.

There exists a table which describes for each information string to be processed all the text qualities which it has, when any basic item is being encountered by any component of the text engine. This part of the environment has to be complete: that is the reason, why we introduced the concept of an entrance point into an information string. This part of the interpretative environment is loaded whenever a particular information string is presented to the text engine.

The second part of the interpretative environment describes not an information string, but the status of the text engine itself: it consists of applicability information for each possible textual quality and optionally a mapping of that quality to an input convention or an output property. This concept can best be interpreted, if we look at the idea of case sensitivity in text operations. Case sensitivity would be modelled in our concept as a state of the interpretative environment, where case is an applicable mode (and characters are during output mapped to different representations).

2.2.2.1 Applicability of Modes

The applicability of a mode consists of a statement, if the status of this mode shall be checked, when a basic item is being encountered by the text engine. If during printing the mode `german` is applicable, a mapping of characters with this mode into another font (or underlining) will be used; if it is not applicable, such mapping will be ignored. *Mutatis mutandis* this is also the case, when comparisons are performed.

2.2.2.2 Applicability of Style

The applicability of style consists of a statement, if it shall be checked when a basic item is being encountered by the text engine. For applicability style could be interpreted like a mode: if it is applicable, any difference in style will make two basic items unequal; there is no way to define a degree of difference.

2.2.2.3 Applicability of Color

The applicability of color defines whether optional I/O mappings shall be used, and a range, within which otherwise identical basic items have to be considered equal with regard to color. This range can be given as a pair of absolute numeric values, or as a table which specifies for each `color` in one information string, which `colors` in the other are acceptable.

2.2.2.4 Applicability of Size

The applicability of size defines whether optional I/O mappings shall be used, and a range, within which otherwise identical basic items have to be considered equal with regard to size. This range can be given as a pair of absolute numeric values, or as percentage of the larger of two basic items, within which a smaller one is still to be considered equal.

2.2.2.5 Applicability of Views

As views are a more general form of the other textual qualities, their applicability is identical to that of the default modes of the four classes given above. The implementation considerations given in 2.2.1.2.5 define the difference between the applicability of any named view of the four types and the four default views of each type.

2.2.3 The Text Engine

What kind of activities our hypothetical text engine is performing, was shown intuitively more or less during the preceding sections. Obviously it is related to the production of output; obviously it is also performing comparison operations: our example of a "german sensitive" comparison as an equivalent to the optional "case sensitivity" of current text processing software implied so much. Though the definition of such a text engine is certainly not part of a discussion of text representation, we would like to include a brief, but somewhat more systematic, definition than given so far. The reason for this is, that we assume, that recent discussions on text representation had a tendency to concentrate a bit too much on printing. We would therefore like to describe operations, we think necessary to make use of all the qualities described in a more general way, i.e., influencing *any* kind of operation the type of text we describe here has to undergo.

Whenever we define in the following sections an ability the "text engine shall have", this is an abbreviated expression therefore for the following reasoning: "We assume that processing historical data of requires a specific ability. If historical data are to be processed by more than one software system, we therefore need a standard for the encoding of the property calling for this ability."

2.2.3.1 Texts Handled

The text engine shall be able to handle texts, which are mixtures of byte coded and bit mapped items. All items a text consists of has to be processable by all components in question. Which class an item has, has to be transparent for any applications programmer using tools provided by the text engine.

2.2.3.2 Import/Export

The text engine shall provide tools to import and export strings. This means, it shall be able to convert its own internal representation of an information string into a form that clearly distinguishes between different classes of items, specifically between byte coded and bit mapped ones, so software components, which do not have the ability to handle both, can extract those portions of a text, which they can handle. This export format, which is described as *external text format* has to be transferable on standard communication links.

2.2.3.3 Comparison and Sorting

The text engine shall have as part of its interface functions which are able to compare and sort sets of information strings, fully controlled by an interpretative environment as described above.

2.2.3.4 Searching

The text engine shall be able to use any information string, which specifically includes such as contain bit mapped items, as a search key in the administration of large string collections, like dictionaries.

2.2.3.5 I/O

The text engine shall be able to convert its internal representation into a form which represents its various classes of items also on output devices, which do not provide means for the most obvious kind of presentation. It also shall contain parsing functions, which convert formats provided by input devices or software supporting sophisticated forms of input into a common internal presentation.