Research

How to Cite: Lepper, Markus, and Baltasar Trancón y Widemann. 2021. “Technical Topologies of Texts.” *Digital Studies/Le champ numérique* 11(1): 1, pp. 1–98. DOI: https://doi.org/10.16995/dscn.313

Published: 13 April 2021

Digital Studies /
Le champ numérique

Lepper, Markus, and Baltasar Trancón y Widemann. 2021. "Technical Topologies of Texts." *Digital Studies/Le champ numérique* 11(1): 1, pp. 1–98. DOI: https://doi.org/10.16995/dscn.313

RESEARCH

Technical Topologies of Texts

Markus Lepper[1] and Baltasar Trancón y Widemann[2]

[1] Erstmittelforscher, DE

[2] semantics GmbH, DE

Corresponding author: Markus Lepper (post@markuslepper.eu)

In Digital Humanities the task of "text modelling" has been recognised and successfully treated in the last decades. But indeed every use of digital text processing software, even the most naive one, is already a kind of text modelling activity. In many realms of daily practice this is executed unknowingly and without theoretic reflection, using digital text processing systems merely as "comfortable typewriters". Then the structure of the out-coming models is determined only by the applied software. To really exploit the benefits of automated text processing in any realm, their use must change to a theory and practice of text modelling.

This requires to explore and make explicit the mathematical structure of the possible text models, and the restrictions imposed on them a priori by the involved technical tools. Those can become crucial especially. when translating a text between two formats – a quite frequent task with surprising pitfalls.

This article gives a systematic and exhaustive survey of the technically determined structural properties of text models. It lists the abstract requirements on modelling tools for ensuring satisfactory flexibility, and compares ten different commonly used text modelling frameworks.

Keywords: Text Processing; Document type; Survey on standards; Mathematical modelling

Les systèmes de traitement de texte numériques s'utilisent, dans la majorité des cas, simplement comme des « machines à écrire confortables », surtout dans les Humanités. Pour profiter véritablement des avantages du traitement de texte automatisé, surtout au niveau conceptuel, l'usage de ces systèmes de traitement doit changer vers une théorie et pratique de modélisation de texte. Pour cela, il faut explorer et rendre explicite la structure mathématique de modèles de texte possibles, ainsi que les restrictions imposées a priori sur ces systèmes par les outils techniques impliqués. Celles-là peuvent devenir essentielles, particulièrement lorsque l'on convertit un texte entre deux formats – ce qui est une tâche très fréquente avec des écueils surprenants. Cet article offre une étude systématique et exhaustive des caractéristiques structurelles de modèles de texte. Il énumère les exigences abstraites d'outils de modélisation nécessaires pour garantir une flexibilité satisfaisante et il compare dix différents cadres de modélisation de texte fréquemment utilisés.

Mots-clés: Traitement de texte; Type de document; Enquête sur les normes; Modélisation mathématique

1 Introduction

1.1 Intention of this Article

In the last three decades there has been enormous progress in philosophical reflection, processing methodology and technical implementation of *digital text modelling.* Most efforts take place in the realm of Digital Humanities or Digital Scholarship and shed new light also on older and fundamental questions about the nature of text and the life-cycles of documents. "We cannot help but realise the great number of domains that inform our understanding of the book." (Siemens, Dobson, et al. 2011) The main task in these realms is to transfer historical texts and documents from a physical medium into the digital sphere, for the benefits of automated administration, access and processing. (Siemens, Dobson, et al. 2011; Pierazzo 2015)

This article starts from the opposite standpoint: Nowadays, even the most simple act of creating a fresh text document with some digital text-processing system is also always an act of "text modelling". Due to the nature of digital technology, even writing a simple "SMS" is indeed the creation of a text model. In contrast to the process in Digital Humanities, this kind of modelling is done unknowingly and without theoretical reflection.

But not only medieval texts, but also contemporary products like cooking recipes, hiking descriptions, usage instructions, poems, novels and scientific texts (like this article) should benefit from automated processing (automated search, indexing, versioning, comparison, evaluation, translation, extraction, montage, etc.) and not at least from *Computer-Aided Reading (CAR).* This term includes all the enhanced or even radical new ways of exploring a text using digital technology, as browsing and searching, referring and indexing, annotating and highlighting, collapsing and expanding, scrolling and animating, etc., – see Section 4.2 below, and Siemens, Dobson, et al. (2011) for a detailed discussion.

To minimise the necessary effort of adapting a text model to this processing pipeline, its nature and structure must be considered. Therefore the attitude of using the computer merely as a somehow more comfortable typewriter must be replaced by the consciousness of creating a text model, which stands at the beginning of an unlimited processing network. This is the first intention of this article.

For computer-based processing different *text processing systems* have been developed, by academic or commercial providers. Whenever a text is created not as the traditional two-dimensional visual representation on paper, but as a data object managed by a particular computer program, then the pre-wired ways of the program's operation, its underlying data categories and their possible relations determine the structure of the created data object. Compared to the pencil on the paper we lose freedom, substantially! To clarify what price we have to pay is the second intention of this article.

Both, the naive users and the skilled specialists from Digital Humanities, often must fall back to the standard tools (from industry or academics) with their pitfalls and idiosyncrasies. These come in play esp. when trying to transfer a particular text from one of these formats into another. To analyse in advance the differences of the possible model structures and the expectable translation problems is the third main goal of this article.

For these goals, we analyse eight digital text formats (**LaTeX**, **Lout**, **DocBook**, **TEI**, **HTML**, **OD-T**, **d2d_gp**, **XML)** plus the two non-digital **Manuscript** and **Typescript** for comparison, see Section 1.5.

Concerning the three goals (automated processing, flexibility of modeling and ease of transfer) our results are mostly *descriptive* about the data models and resulting restrictions. When a particular system allows widely varying ways of usage, the text may become *prescriptive*, recommending one way of usage over the other.

1.2 Structure of this Article

The rest of this section clarifies the basic notions and principles for the further discussion.

The next section discusses *global properties* of the different tools, which apply to a text as a whole or to the general treatment of text data.

Section 3 analyses the treatment of the different text components as defined at the start of that section and visible in the headlines of the subsections.

Section 4 treats dynamic/temporal aspects of text modelling.

Section 5 finally collects all found differences into survey tables.

1.3 Fundamental Definitions

"In most all things that exist at the intersection of several domains, domain-specific cultures have potential to collide, in useful ways as well as others." (Siemens, Dobson, et al. 2011) This article operates on the borderlines of computer technology, informatics, general humanities, linguistics, philosophy, etc. To collide in a useful way (and not in others!-) it seems necessary to first clarify the nomenclature, because the central words needed in the following are used in these different domains with different meaning, and maybe even in some of them alone. We have to define "text", "rendering", "model/modelling", "text modelling framework", "substance", "accidentals" and "identity". Notabene, the following definitions shall only serve the following concrete discussion, – naturally we do neither intend nor expect to give finite answers to highly controversial questions in the various domains.

"The answer to the question what a text *is* depends on the context, methods and purpose of our investigation." (Huitfeldt 1994, pg.235, our emphasis) In the following discussion, the word *text* is used for an *intentional object* in the sense of Husserl (Jacob 2019). Thus our "text" is nearly identical with the "document" defined as an "abstract object" by Renear and Wickett (2009, 2010) and to the definition used by Ingarden (1960) in his aesthetic analyses.

As an intentional object it is a merely mental, psycho-internal symbol which is completely empty: It has no properties at all except definedness and self-identity. It must be (explicitly) related to some outer material objects as its representations, for becoming communicable. Such a material object is called *rendering* in the following. It corresponds to what Huitfeldt, Vitali, and Peroni (2012) call "manifestation of the (same) document" and Buzzetti (2009) calls "image" ("the text does not have a material nature"; "'the text is only [and] always an image") and what Pierazzo (2015) calls "document". Further our "text" and "rendering" correspond to what FRBR calls "work" and "manifestation" (IFLA Study Group on the Functional Requirements for Bibliographic Records 1998). Physical objects like coloured lines on a piece of paper, waves of sound when hearing a lecture or bits and bytes stored in a computer system are different renderings of the same abstract text object.

A *model* is a special kind of rendering, as it is *structured*. Its information contents are segregated into distinct pieces, called *(model) elements*. Each such element is assigned one particular *type*. Each type prescribes the possible relations of its instances to the instances of the same or other types.

The definition of all types and their possible relations is called a *meta-model*. Every meta-model defines an infinite set of all possible adherent models. Every software tool and every text format definition comes with its own and specific meta-model. This article thus compares different meta-models.

With these definitions, the dots of ink on a sheet of paper are possibly the rendering of a text, but not a model. (But of course the *contents* of the text can *describe* a model, as in our **Figure 1**.)

What humans do when reading the text from such a piece of paper is creating an *inner, mental model*: Pixels of colour are translated into elements of types "headline", "caption", "footnote" and their respective relations. These types of (technical) elements which are well-known to the author and the elements of which are handled explicitly are also called *text components* in the following, esp. in Section 3.

The same holds with computer data: A JPEG file with a photograph of the paper and its dots of ink is a computer based rendering but not a model. Pierazzo (2015, pg.33) calls this "only digitized", in contrast to "digitalised". Contrarily, the computer based text processing programs discussed in this article work on structured data, means: on models. Each of them comes with its specific meta-model and is therefore called a *text modelling framework (TMF)*.

The very first step when using a TMF (the transfer of a given text into this TMF, or the creation from scratch of a new text) already has significant impact on the structure of the model, because this has to adhere to the TMF's meta-model: A mosaic of the Mona Lisa will appear differently when the available tessera are of three colours and rectangular or of 273 colours and hexagonal.

In this concern, very different software products like the (abstract) encoding specifications TEI/XML and the (concrete) layout processor LaTeX stand indeed on the very same level: They both try to catch the substance of a text into a model, – a first and already transforming step, independent of all further processing and intention. Only this step and its target meta-model are subject of this article.

By the way: This notion of "model" must not be mixed up with the various ways for catching the *contents* of a text, i.e. modelling all the persons, things and facts mentioned *in* the text into a network of model elements, possibly represented as digital data. This is the subject of the very different variants of "Computer Assisted Qualitative Data Analysis Software (CAQDAS)", creating something we could call "contents topology". This is totally disjoint from the "text models" meant in this article, which refer to the text *as such*, as a mere object of language or writing. Here the model element types are "caption", "headline" and "footnote", forming a mere "technical topology", as indicated in this article's title.

Internally a postscript/PDF file is also a model. But its element types are again totally different, namely "fonts", "subroutines", "graphic contexts", "coordinates", etc. In relation to text this is only a rendering, not a model.

Being an intentional object, the relation from the inner mental symbol to an external rendering is always established *explicitly*, by an intentional act. This relation is personal and explicit, and whether a particular material object is related to the abstract text identity as its rendering is always a matter of personal opinion and taste. A frequent counter-example are statements like "Not, this *was not* the Moonlight Sonata, because the first movement was much too fast."(Ingarden 1962, pg.126)

In our context, this intentional act becomes relevant when transferring a given model of a text from one rendering into a different meta-model: Every aspect which is considered *substantial* must be preserved, and everything considered *accidental* can be omitted, or even: must be omitted, because it is not supported in the target meta-model.

Based on this dichotomy also a notion of *identity* can be constructed: Two renderings realise "the same" text, i.e. the texts realised by them "are identical", when the *substantial aspects* have the same value in both renderings. So the identity of a text is established by its substance, while substance and accidentals are both required for human reading and writing.

But there are no a priori or general rules to classify a particular aspect as substantial or as accidental. "The choice of what to include is a crucial one, and depends mainly on the purpose of the edition [/rendering]". (Pierazzo 2015, pg.39)

In most cases it will be understood that the concrete *optical appearance* (e.g. the fonts and font size selected by the publisher) belongs to the accidentals, not to the substantials of the text and does not establish a different identity, see Section 1.7 below.

Contrarily, it *is* substantial that a part of a sentence is printed in an "emphasised" way, independently how this emphasis is realized (by bold type, underline, colour change or sim.) or that a sequence of words is printed "as a section title". In modern terminology: The *physical mark-up* is translated into some *semantic mark-up* before two texts are compared for identity.

Furthermore, for many classes of texts it is agreed upon that *line breaks* and *page breaks* are not part of the substance. But there are classes which do respect line breaks, like poems and theatre plays in verses. It may be even considered of significance whether a poem appearing in a novel is separated by the preceding flow text by zero(0), one(1) or two(2) blank lines.

In the documentation texts of the different TMFs discussed in this article, those of **TEI** are the only ones which explicitly discuss the problem of text identity, substance and accidentals. A standard publication says e.g. "[…] one aspect of a fundamental encoding issue: that of selectivity. An encoding makes explicit only those textual features of importance to the encoder." (Burnard and Sperberg-McQueen 2012), and another: "These broad definitions of textual aspects are not necessarily exhaustive and can be expanded and tailored to cover particular research aims and textual phenomena. Indeed, potentially there are infinite ways of making, describing and interpreting models of texts." (TEI-Consortium 2013, sec.3.4) (In these two quotes, "encodings" and "models" seem to be used exchangeably.)

1.4 Analytical Tools From Mathematics and Compiler Construction

This article tries to systematise our experiences with modern, scientific and fictional texts and with their automated processing, from the standpoint of *compiler construction*. This is a discipline in informatics which originally has dealt with very special texts, namely those written in programming languages. But it also has developed a formal perspective on text in general, as a necessary pre-requisite for algorithmic processing. So any language becomes a kind of "computer language" as soon as it is processable by an algorithm. As a consequence, some fundamental principles of compiler construction can be taken over when talking about digital text processing in general.

One of the most fundamental and beneficial principles in software design is *compositionality*. This means that features or behaviours can be applied in arbitrary chains, where each step can get as its input the output of (a) *any kind* of preceding step, and (b) needs to know nothing about its application context.

Of course there are particular settings where compositionality cannot hold. But then it must be *restricted explicitly*, and this can be a hard criterion for the kind of text model. E.g., the restriction "Text in title lines may not contain footnotes." is apparently true in most "plain" texts, but not when the footnotes are annotations by the editors in a historic-critical edition. For more details see Section 2.6 below.

Another every-day principle is the distinction between a *foreground representation* (also called *external representation*), which lives in one particular rendering of the model, versus the *middleground information*, which is the *model in the narrow sense*. E.g., in many computer languages an external object can be referred to from a source file by a *qualified name*, which is a chain of identifiers, joined by colons or similar punctuation. Alternatively, only the very last of these identifiers must be written, if its containing "module" has been "imported" before. Both are different front-end phenomena, but the information content (for further processing) is totally identical. Similar techniques happen in natural language texts quite frequently, but in most cases are not made explicit, – see Section 3.5 for important cases.

Related principles are *separation of concerns (SoC)* and *minimality*. The former means that fundamentally *different* aspects of things of one particular kind should not be mangled into one syntactic form, but should be modelled by one different syntactic element each. This allows to *re-use* these front-end forms to model the same aspect of "things of total different kinds", and leads to minimality of different syntactic forms. This requirement is often violated, see Section 2.5.

1.5 Compared Text Modelling Frameworks (TMFs)

In the following sections these TMFs are compared:

- **Manuscript**
  Text written by hand.
- **Typescript**
  Text written using a typewriter.

- **LaTeX**
  The well-known type setting system by Leslie Lamport, based on the TeX system by Donald Knuth. (Lamport 1986; Goossens and Mittelbach 2004)
- **Lout**
  A compiler front-end to generate postscript output, by Jeffrey H. Kingston. (Kingston 1992, 2000b, 2000a, 2013)
- **DocBook**
  An XML-based (originally SGML-based) framework for technical documentation, esp. for computing technology, chief designer is Norman Walsh. (Walsh 2010; DocBook-Team 2014)
- **TEI**
  "Text Encoding Initiative", a project for standardizing the encoding of arbitrary texts, but esp. in humanities. We speak about the latest version "P5", which is not longer based on SGML, but on XML (Jannidis 1997; Branden, Terras, and Vanhoutte 2008; TEI-Consortium 2016, 2013).
- **HTML**
  "Hypertext Mark-Up Language" is the original document encoding of the world wide web, started by Tim Berners-Lee. Looking at the contemporary state of the art, we mean "XHTML 1.0", and "Cascading Style Sheets CSS 2.0" will be taken into account whenever relevant for structural features.(W3C HTML Working Group 2002; W3C HTML Working Group 2011)
- **OD-T**
  "Open Document - Text" means all parts of the Open Document OASIS standard which are relevant for text documents. (Durusau and Brauer 2011)
- **d2d_gp**
  "Direct Document Denotation – General Purpose Module", an front-end for writing down XML encoded documents with least possible formal noise, in the flow of creative authoring, by the authors. (Lepper, Trancón y Widemann, and Wieland 2001; Trancón y Widemann and Lepper 2010)

- **XML**
  Sometimes a feature applies to all XML-based frameworks, i.e. **DocBook, TEI, HTML, OD-T, d2d_gp**, and potentially others. (Bray, Paoli, et al. 2006; Boyer 2001)

Since we claim the change to "text modelling instead of type writing" as unavoidable and beneficiary, we do not speak about the very first solutions like `roff` and TeX, because they realise only the visual aspects. Nevertheless they are worth mentioning, – without the experiences they brought, none of the TMFs listed above is thinkable.

Also **postscript** (Adobe Systems 1999) and PDF (Geschke and Warnock 2006) are no primary subject of analysis in the following, because, as explained above, they serve as mere rendering targets and not as meta-models. Nevertheless some of their technical properties will become relevant when discussing particular tool chains.

What severely limits the values of comparisons is the fact that all these systems are more or less parametrisable, extendable and adaptable. E.g. LaTeX is a Turing complete programming language, – every thinkable modification can be implemented. With others of these frameworks, the variability is more restricted, or maybe very hard to define formally, but always given to a certain extent.

Therefore the following comparisons are deliberately restricted to the *off-the-shelf, unmodified* state. Comparing them thus gives an impression of the different basic philosophies, – but to realistically judge the costs necessary e.g. for translating between these models, the costs of adaption and parametrisation of the target meta-model can be significant. Nevertheless, in few cases when some particular parametrisation is especially easy or even required, this will be mentioned in the comparison results.

1.6 Substance and Identity, Revisited

Now that the collection of TMFs under consideration has been defined, this is how they treat the above-mentioned question for substance and identity of their text models:

Manuscript and **Typescript** have a comparable simple starting point for the questions of substance and identity, since there is only one physical exemplar. In pre-digital times, **Manuscript** and **Typescript** have been the standard format for authoring in the sense of "writing down", but also for corrections, re-arrangements, "working on" a text. And finally for communication between author, editor, type setter. Sometimes these phases are multiply interspersed and thus different temporal layers of writing and erasing form a *temporal sequence of different text bodies*, each being derived from its temporal predecessor.
So with **Manuscript** and **Typescript** the answer to the question for identity can be the most simple of all TMFs, but may also turn out most complex, as discussed in detail in Section 4.3.
A further aspect of the "text as such", as it shall be encoded, reproduced and processed, it the common practice to use *abbreviations*, which are "meant to be expanded". So the *expanded text* can be considered to define the identity.
By the way: the English language orthography clearly shows the difference between the two cases that (a) an abbreviation is only a denotation in the external representation, standing for its expansion in the the model, or (b) the abbreviation itself, as such, is part of the text model. Eg. with XML entities, correct sentences are (a) "she is a &MP;", as shorthand notation for "she is a Member of Parliament", vs. "she is an MP".

LaTeX and **Lout**: The text contents and the source text are two very different kinds of text, as with any compiler language, see above. Additionally, the generated Postscript output is a third kind of text on its own. Identity can be discussed on all these levels.

DocBook has as a ruling idea the *single source* philosophy: printed manuals in different formats, online screens, even interactive help pop-ups in applications, shall be derived from the one single DocBook document. From this point of view, this is the "real" document, ruling identity and equality, and all derived artefacts are "just renderings".

TEI is the only context where the problem of text identity, substance and ontology is discussed explicitly, see the quotes at the end of Section 1.3 from Burnard and Sperberg-McQueen (2012) and TEI-Consortium (2013, sec.3.4). So the notion of "identity" is (at least implicitly) recognised as being critical.

d2d_gp defines no special diff-like mechanisms beyond those for **XML** in general.

XML in general: A mechanical device applicable to all XML-based encodings (**HTML, OD-T, DocBook, d2d_gp**) is *canonical XML*, which unifies tag names, white space, formatting, etc., and thus makes the simple unix standard tool "`diff`" applicable. (Boyer 2001)

1.7 Exclusion of Optical Appearance

The "substance of a text model" defined in the preceding sections seems sufficient for all kinds of texts, scientific or fictional. But indeed it imposes a severe restriction, which can turn out unacceptable in other contexts: The *optical appearance* of the text is excluded from the model's substance; the concrete graphical layout only serves as a means for representation, as a carrier for the meant contents; it is accidental, not substantial.

This is inadequate for many products of *fine arts*, starting with ancient stone engravings, designed in equal rights as text message and as graphical ornament, ranging over many kinds of medieval manuscripts, up to the Concrete Poetry and Dadaistic poems of the early twentieth century: In her standard textbook, Pierazzo (2015)[pg.51] quotes a "calligramme" graphic by Apollinaire. In all these examples the concrete graphical appearance is *part of the meant substance*, not only a necessity for transportation.

A more recent example, fitting into the discussion of digital TMFs, are the poems of Mauricio Rosenmann, which use the $\TeX$ typesetting system for rendering words and sentences according to aesthetic considerations. Font style, size and position of the characters being substantial part of the message (Rosenmann 1995, 1996).

This wider definition of a text model is *not covered* in the this article, but nearly all categories described in the following sections could be enhanced accordingly.

2 General Properties of TMFs

In this section we treat properties and strategies applied by the different TMFs to a text as a whole.

2.1 Multi-Layers and Multi-Authors Texts

A text often is a homogeneous object we look at simply for reading and grasping some contents. But this is only the simple case. Indeed, a text can be a multi-layer object: an original text can serve as the basis on which later altering operations have been performed. To represent this layering is obviously something different than to model only its results; the former has obviously at least one (≥1) dimension more.

So the text of a poem by Hölderlin (to take an extreme example) is for the readers in their rocking chair one consistent speech they want to listen to and meditate about. But for the philologist it is a stack of corrections, alternatives, strike-throughs and rub-outs, reflecting a fascinating process of word finding.

Already a simple critical edition of some text T1 of some author A is indeed an aggregation of *two(2)* very different texts, when applying the preceding definitions and considering authorship: First the inner text, the edited work by A forms a text. There is a second text T2 written by the editor(s) E (E1, E2, …), which completely fills the appendices. Additionally there are at least the annotation marks, interspersed into the main body but pointing into the appendix, which are part of T2 but physically rendered in the flow of T1.

T2 is a text on its own, but is only readable together with T1. So for the reader the two texts presented are T1 alone and T1+T2 together. T1+T2 is an achievement of the editors; of T2 they are also the authors.

T1 and T2 are called *substantial layers*, because the questions for substance and equality must be answered separately for both. In a similar way, different layers of typescript, first manual corrections, second layer of corrections, etc., form different substantial layers.

When the editor *inserts reconstructed text* into the flow text of the main body, the definition becomes ambiguous: The reader probably will receive it as "one particular version of the text T1 by A", but with the same right it may be said that the completed sentence does only appear in T1+T2.

In case that page break indications of some earlier reference edition are interspersed in the rendering of T1, then these are part of T2, not of "the substance of T1".

Similar: in many cultures the print versions of laws, or of books of the Bible, contain "interspersed local headlines" before each single paragraphs. These shall serve for quick orientation and are not part of T1 but of T2. Since this is well-known, they can come without further mark-up.

Summarising all these aspects, it seems clear that there cannot be a general a priori definition of substance and accidentals of anything called "text". Instead, the notion of text and its identity must always be discussed *explicitly and anew* for each modelling project. Otherwise later automated processing will yield unexpected results.

2.2 Meant Model vs. Written Source Text (SrcTS)

A basic property of a TMF is called *Source Text Strategy (SrcTS)*. It means that there are two different substantial layers of text: first there is (A) a *source text*, which is a "physical object", which is stored in a file system and which can be printed, read and edited like a program source text. The author operates on this text. A very different thing is (B) the *intended text model*, which is described/constructed by evaluating (A) according to the rules of the TMF. (Actually, SrcTS is a property of nearly all digital TMFs discussed here, except **OD-T**.)

Indeed, both layers of text are text formats in their own rights. The source can be regarded a kind of rendering on its own, namely an especially ugly one. It happens frequently that a reviewer's comment about a submission to a mathematical conference contains a fragment like

```
... the case a\not\in\alpha should be discussed. ...
```

which uses the non-rendered **LaTeX** *source* text for communication.

In many TMFs, the difference between both text layers can vary with the context: source text sections which are marked as "`verbatim`" (or "`source`" or "`screenshot`", etc.) will transfer their whitespace and newline characters from source text into the text model and thus into the rendered output. But in text sections of most other kinds these characters belong to the source text only and are not part of the model. The same difference can be found between "physical whitespace" in the source, and that which is wrapped in a `<text:s>` element in **OD-T** a kind of "non-ignorable whitespace", (Durusau and Brauer 2011, sec.6.1.3) or in a `<xslt:text>` element in XSL-T.

The mere existence of a source text does not help when searching for the real substance of a text. This can only be found in the mathematical model, not in the concrete front-end grammar of the input language. Here again the fundamental difference between foreground-representation and middleground model rules, as mentioned above. Some framework languages are even Turing complete, i.e. full-fledged programming languages, and there is no way of calculating the equality of two sources (=the identity of the encoded models) without their complete evaluation.

But frequent practice does help to narrow down substance considerably, as the source is often stored in two disjoint sets of text files. One contains the "*style sheet*" sources, which are provided by the publisher and thus accidental. The other contains the "*contribution*". All definitions therein are more likely to be substantial. (This practice is ruled by the principles of SoC and reusability, see Section 2.5.) Any re-definitions in the latter of rendering rules from the former are also candidates for substance. These considerations apply to SrcTS TMFs, i.e. compiler style text models.

A further step of formalisation perform the TMFs with an *XML encoded source text*. These are **DocBook, TEI, HTML** and **d2d_gp**. This allows to apply standard tools and fine granular access control, see e.g. Section 4.4 below. (**OD-T** has also an XML based representation, which is normally not edited directly, but which can also serve as starting point for automated processing.)

An important consequence of SrcTS, that all kinds of meta-information (like "Fixmes", "todo lists", comments on open issues and possible variants, notes about the calendric dates of changes and updates) can be reified and managed in the same document as the definition of the intended text model, namely as source language level *comments*. This can be esp. useful for multiple authors' co-operation, and has the additional effect that automated retrieval by version control systems, automated comparing, searching and replacing (`diff, grep, sed,` etc.) apply to these meta-info seamlessly in the same way as to the text model itself.

LaTeX, Lout, DocBook, HTML, d2d_gp and generally all **XML** based TMFs follow the source text strategy and thus can make use of *comments* in the source text. (The same holds indeed for **postscript**, but since this is only qualified as a *back-end*, these comments cannot be used by authors. Indeed, many software which generates **postscript** does insert comments to document their operation, or even for further processing.)

OD-T also has an **XML** based output format, but this is not useful for manual writing (authoring), only as a program back-end.

2.3 Source Model Coupling (SrcMC)

In meta-models which are based on SrcTS, most definitions of the semantics of input structures follow the principle of *source model coupling (SrcMC)*. This means that the sequential order of the sub-expressions in a compound expression in the source text directly specifies the sequential order of the sub-elements in the model.

This is trivial and understood for the words in a sentence, the sentences is a paragraph, etc. But it is not always convenient for cells of a table (see Section 3.3.2), for footnote text (see Section 3.7), for defining index positions (see Section 3.5), etc.

For example, **LaTeX** supports two flavours for footnotes: mostly the complete source text of a footnote is written "in place" after that portion of main text which shall be decorated with the footnote mark. So source text and model are coupled (= footnote-SrcMC).

A long footnote text in the middle of a sentence lowers the readability of the source text, so **LaTeX** allows as an alterative to write two(2) separate expressions in the source: a short position indication in place, in the middle of the sentence, and the footnote text separately, some lines later. The meanings are exactly the same with both methods. So again the source text does not help to decide the equality of the intended model substance.

2.4 Compound Source Strategy (CmpSS)

Supporting the *Compound Source Strategy (CmpSS)* means that one single source file can contain segments of very different text description languages: each of these will be translated by different dedicated compilers, and the results will be combined to make up the final document rendering.

The advantage of the compound source strategy is, that all information is kept in one single source file (or a sequence of source files, organised according to the user's need), without fragmentations due to mere technical necessities. This kind of integrity can also be aimed in conventional settings, with different files, by collecting them in one single disk directory, or one single `zip` archive, etc. But then still losses of single physical files may happen, or some confusion of file versions, and the integrity of the "text as such" will be lost.

(By the way: CmpSS can be seen as a successor of the ancient "*object linking and embedding (OLE)*" technology by microsoft. The fundamental difference is that there *binary* objects have been linked, and only dedicated computer software could perform the embedding and processing, while our notion of CmpSS means concatenating human readable source texts.)

A second advantage is, that all search, replace and spell check operations are applied on the whole text model, seamlessly and consistently, when applied to the one single source file. For instance, the replacement of one name by another will happen in figures, song lyrics, graphics and in the main text consistently.

Manuscript and **Typescript** follow CmpSS in so far as graphics, charts, etc., drawn by hand or by type writer can be integrated in any document.

LaTeX supports CmpSS when "`pgf`" or "`pstricks`" or other graphic compiler packages are used (Tantau 2015). Similar with "`musixtex`": replace operations by an editing program on the source text will affect the normal flow text and e.g. song lyrics in the same way and synchronously. The same holds for bar numbers, etc. (Taupin, Mitchell, and Egler 2002)

Lout follows CmpSS for graphics in general, and some specially supported kinds of diagrams (graphs, charts, syntax diagrams, pie graphs). (Kingston 2013, pg.165pp)

DocBook does support CmpSS only so far as "screen shots", "program sources" and mathematical formulas are supported. All other "media objects" are defined externally and embedded in an opaque way. (Walsh 2010, sec.3.6.8).

TEI comes with its own document type definition language "ODD" (TEI-Consortium 2016). This allows to integrate third-party elements, so basically CmpSS can be realized according to the needs of a particular encoding project.

HTML follows CmpSS as there are combined formats of "`HTML+SVG`" or "`HTML+MathML`", which allow graphics and mathematics formulas to be included. (But severe technical problems arise, since these combinations are not covered by standard DTDs.)(W3C 2002)

OD-T in principle aims at CmpSS since a file declared to be an `<office:text>` document may contain also element types from the graphics, the charts, the spreadsheet world. But it seems that this transparency has not been realised completely, but put under rather idiosyncratic limits. (Durusau and Brauer 2011, sec.3.5-3.8)

d2d_gp supports CmpSS as first class resident: the `#embed` operator is foreseen to escape to arbitrary text based compilers, and currently extensively used for the inclusion of `LilyPond` source texts (Lepper 2015).
(E.g. the sources of an SVG text or an `xfig` object are pure text, thus can be embedded into **d2d_gp**. A future d2d-aware editor should extract these fragments automatically, start an external editing application, and finally re-embed the result.)

XML in general has as one of its fundamental design goals the *free combinations of different formalisms.* Therefore it is a natural basis for CmpSS.

2.5 Separation of Concerns (SoC), Minimality, Reusability

A pre-requisite for reusability of styles, layout methods and structure definitions is *separation of concerns (SoC)*. (It is also a pre-requisite for compositionality, because for re-combining aspects you first have to separate them !-)

SoC means that different aspects of the text model and its renderings are realised by different data structures, notated in different parts of the source text, or even in different disk files.

Most TMFs support this strategy. Nevertheless, many implementations of tools and applications counteract the intended effect. E.g., **OD-T** separates the rendering information for segments of character data from their contents, and allows one "rendering style" to be applied to a multitude of text segments by named references. The number of styles should be minimised and styles should be re-used in a sensible way using the inheritance mechanism provided. But the wide-spread open source implementation "libreOffice" (3.5:build-413) applied to such a sensibly designed text model, does *expand* these definitions before writing them out and thus creates a new style object for each single character segment. This turns the feature of separate, re-used and named style definitions from a means for clarification into a means of confusion.

A special instance of SoC is the separation of *semantic mark-up* and optical appearance. As follows from our basic definitions of text, see Section 1.3 above, this is indispensable.

Manuscript and **Typescript**: (n.a.)

LaTeX has been developed as a consequence of the SoC principle: it is a collection of TeX programs which encapsulate requirements for particular standard text formats (i.e. TeX applications) into so-called `class` files. The realisation of these formats is full-fledged, Turing complete code, which finally calls TeXs original type setting API for producing output (Lamport 1986).
In a next step, the user can tailor those formats again by further macro coding. These can be collected into further modules, realised as module include files, or even as "class files" on their own.

Lout follows basically the same architecture als **LaTeX**, with different names for modules and code files.

DocBook has a monolithic architecture, tailored to the specific needs of technical documentation, esp. computer software. Adaption of its definitions, thus re-use, is explicitly disencouraged (Walsh 2010, sec.5).
SoC is e.g. violated by defining `<mediaobject>` and `<inlinemediaobject>` as two different element types with nearly identical structure. This practice is notorious. Another negative example is `<info>`, which is a common wrapper for very different, unrelated kinds of meta information (sec.3.5), and the three distinct element definitions for paragraphs. (sec.3.6.6)

In **TEI**, SoC as a first class principle induces the highly modular structure of its document type definitions.

HTML supports the separation of semantic mark-up and rendering, i.e. optical appearance. A `@class` attribute may be applied to the text body's elements, according to semantic roles, and these classes linked to final physical rendering parameters by some CSS text in dedicated, reusable files.
Furthermore, dynamic behaviour realised by ECMA script code can be (and normally is) put into separate files for SoC and reusability.

OD-T aims at SoC by separating several kinds of "style definitions" (which are basically rendering parameters) from the real text model object. But the design and its documentation are more confusing than clear, and seem heavily redundant, thus missing SoC (Durusau and Brauer 2011, sec.3.15, sec.16).

d2d_gp tries to realise compositionality, SoC and re-use as far as possible. It is the only TMF which employs the versatile technique of *parametrisation by rewriting*, which allows to change any detail when instantiating modules and adopting them to different application contexts (Lepper and Trancón y Widemann 2018).

It makes any existing packages parameterisable in an utmost flexible way, by a simple input language very close to the user's intuition. Contrarily, the necessary fix point algorithm needed for evaluation turned out as rather complex, what seems the reason that this approach has not been used before.

XML in general has as one of its fundamental design goals the free combinations of different formalisms. Therefore it is a natural basis for SoC.

2.6 Compositionality

As mentioned in the beginning (see Section 1.4) compositionality is one of the most beneficent principles in software design. It basically means two properties, or one property seen from two sides, namely (a) to allow the free combination of all types of components, as described so far, and (b) to treat each component according to its type in a uniform way, independent of its enclosing context. Compositional behaviour should always be the default case: the human user as well as the author of the program code do profit heavily.

Contrarily, a particular TMF may impose *explicit restrictions on compositionality*. Some of them are sensible, commonly accepted and beneficent, e.g. in favour of the user's orientation: Many TMFs have a structure like "*paragraph*", which cannot directly be used recursively (a paragraph cannot contain paragraphs), but indirectly (a paragraph can contain tables, which again can contain paragraphs). So the user can make their mental model always as a "simple one-dimensional chain of paragraphs", with complicated structure above and below. More restrictions are wide-spread, e.g. footnote text cannot contain footnote marks. All these restrictions must be made explicit, when discussing the text structure, and must be sensible w.r.t. the goals of modelling.

In many TMFs there is a quite different group of restrictions which come only from technical implementation problems, or from bad design of the standard, and are not related to the modelling problem as such. In most such cases other design goals like separation of concerns and minimality are also violated.

Without restrictions on compositionality, lists can contain lists and tables and paragraphs; tables can contain lists and tables and paragraphs; paragraphs, whether in lists or tables or both or neither, can contain annotation marks; annotation text can contain tables and lists and lists of tables, etc.

So far no problem should arise, only some questions. E.g., where is the footnote text placed if the footnote label is in a table entry? At the bottom of the table or the bottom of the page? And what about annotations in a float caption?

A systematic solution is to assign to each annotation kind two lists of component types and kinds, the innermost component from the first (/second) list which contains the annotation mark defines the reset scope (/rendering position).

In most cases a title text needs to contain entity references and formulas, sometimes also highlightings. But does it need citation keys? Or footnote marks? And how will conflicts of the clashing rendering rules will be resolved?

In most cases title text will explicitly exclude two dimensional components like lists, tables and diagrams. But possibly float captions may have richer contents than section titles have.

In most cases annotation text (e.g. footnote text) may contain everything like a normal main text paragraph. But annotation marks are limited, as discussed in Section 3.7.

W.r.t. entity references and annotation marks a new aspect comes into play, namely the multiple rendering of one source text segment (what can be called *multiplication of source text*): A section title or float caption text appears in the rendered text document in the component it belongs to (the section title paragraph, etc.), and additionally in a central index, ("ToC" or "List of Figures"). Anchors, citations, entity references may only be "operative" (rendered as a link target or added to a register) in the first case, not in the second.

In any case, the adequate means for a mathematical model of these restrictions are restrictions on *the transitive closure of the containment relation.* Please note that this is a rather high-level device, even pure relational algebra does not suffice. It must be formulated additionally to the grammar rules of "content models"; most attempts to integrate it therein are highly impractical and hardly maintainable.

E.g. the original SGML design has a mechanism for forbidding particular (direct or indirect) nestings. Obviously due to lack of practicability this has been dropped in the definition of XML.

In **Manuscript** and **Typescript**, authors are a priori totally free to arrange the text as they like, without restriction on compositionality. This can lead to a multi-layered and even ambiguous text structure, which complicates the transfer into digital meta-models (and thus automated processing) substantially.

LaTeX and its basis TeX have been explicitly designed for compositionality. Due to the functional flavour of the language, there are no fundamental limits for nesting: footnote text may contain footnotes, lists, tables and paragraphs may appear in any order, etc.
Nevertheless, there are severe limits. Partly by intention: floats may not contain floats nor chapters, sections only subsections, not subsubsections, etc.
Others limitations are mere technically induced, mostly by the use of global variables. E.g. all structural levels are realised by the *names* of variables, which makes context-independent programming tedious and inefficient.
Nevertheless it is amazing that most classes and packages of the **LaTeX** world can be combined and plugged together, – in thirty years of experience, using packages from very different realms, we encountered two or three name clashes and two or three fundamental incompatibilities!

Lout is, w.r.t. compositionality, again similar to **LaTeX**.

DocBook partly aims at compositionality. The elements with explicit nesting level `<sect1>, <sect2>` ... behave non-compositional, as in other TMFs. But there is a compositional element `<section>` as an alternative. Top-level structure elements are not compositional (same in other TMFs). (Walsh 2010, sec.3.4). But in most of its definitions, compositionality and SoC are completely negated.

TEI basically supports compositionality by its module mechanism. But since all changes in imported modules must be defined pointwise and separate, integration of user defined modules may turn out to be tedious. A bad example for missing compositionality and confusing documentation can be found in a major text of the TEI-Consortium (2016, sect.16.3/pg.536):
"In other cases, [...] the `<p>` element may be used to tag the paragraphs, and the `<seg>` element used to subdivide them. The `<ab>` element is provided as an alternative to the `<p>` element; it may not be used within paragraphs. The `<seg>` element, by contrast, may appear only within and not between paragraphs (or anonymous block elements). "

HTML basically aims at compositionality. Even hierarchical levels of chapters and sections do really mean only the optical rendering of their headlines and are in no way restricted (see Section 3.1). But some decisions are hard to accept, and the work-arounds hard to understand: lists "`<ol>`" cannot be contained directly in paragraphs "`<p>`", but wrapping them in a "`<button>`" or "`<ins>`" works!? (Trancón y Widemann and Lepper 2019)

OD-T The basic treatment of "style definitions" seems compositional, since the combination rules allow to resolve any conflict. (Durusau and Brauer 2011, sec.3.15, 16) The conflict resolution rules are sketched out in natural language, which is rather confusing. (sec.16.3)
The text body components have very similar restrictions on compositionality as known from the oldest predecessors (e.g. paragraphs cannot contain paragraphs, but tables can contain tables, etc.)

d2d_gp: (see remark on SoC above)

XML in general has as one of its fundamental design goals the free combinations of different formalisms. Therefore it is a natural basis for compositionality.

2.7 Tree Structure and Free and Bound Segments

Historically starting with SGML, all technical text meta-models listed in the previous section and treated in this article, except manuscript and typescript, are based on a structure with the mathematical definition of a *tree-shaped graph with directed and ordered vertices*. In most implementations this graph is technically reified as a so-called *Document Object Model (DOM)*. The nodes of that graph are either so-called *elements* or *character data nodes*. A text is modelled as one(1) top element; each element has an ordered final sequence of *child nodes* and a set of string values, indexed by string values, as its *attributes*. The element graph is free of cycles. Nowadays this structure is omnipresent in XML encoded text objects. But since this is a natural encoding of *expressions* (in the mathematical sense), also other formats like **LaTeX** and **Lout** basically adhere to a tree structure.

Of course this tree form appears a natural and direct solution for many aspects of text, e.g. the hierarchical structure of an article with chapters, sections, appendices on top, paragraphs below, and words, whitespace, symbols and icons as *leaf nodes*.

But indeed the tree form is *not* an adequate model for text as such, not for all of its aspects. E.g., many of the TMFs can model a selected text segment (=a contiguous sequence of characters, subject of a recent text change or of an annotation) only by wrapping it into one(1) element, i.e. making it the contents of one(1) node of the tree structure, see the dedicated elements of type `<sc>` = "select contents" in the lower part of **Figure 1**. This is called *tree-bound (text) segment* in the following.

But many practical use cases are not covered by this structure: the **manuscript** TMF has no problem to highlight arbitrary text segments with a text marker, starting in the midst of one paragraph and covering also parts of its follower.

In a tree like base structure, this can be modelled only by *two(2)* elements, one representing the start and the other the end of the text segment. These two elements have themselves empty content and are related to each other by some identifier value, see the empty elements `<ss/>` = "start segment" and `<es/>` = "end segment" in the lower part of **Figure 1**. This pair is called a *tree-free (text) segment* in the following. (The most important application of this principle will be discussed below in Section 3.4.)

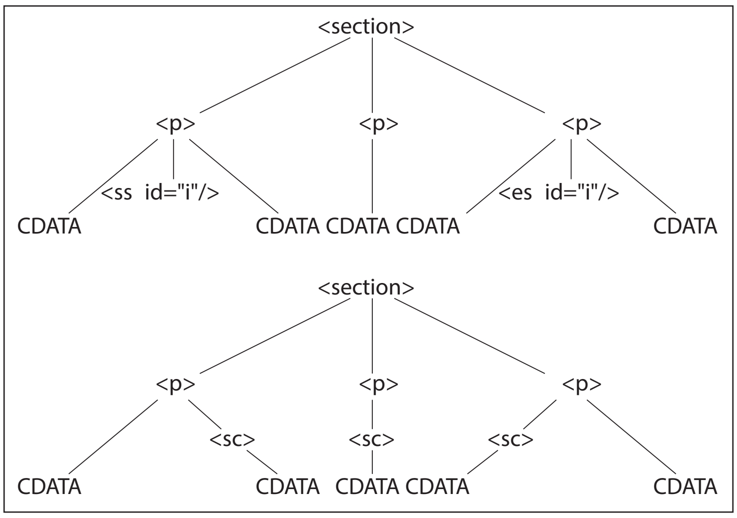

Figure 1: Tree-Free and Tree-Bound Modelling Of the Same Character Segment.

These tree-free segments by elements can be positioned independently from the tree structure; nevertheless their element types must be foreseen explicitly in the document type definition of each element type they shall be contained in. One main technical disadvantage is that their correct pairing is not implied by the syntactic document type, as it is with tree-bound text segments, but is a *semantic* property, requiring explicit validation.

Even more important is that tree-free segments are not only independent from the tree structure, but also from each other: they need not be nested properly but can overlap arbitrarily, like different text marker applications in a **manuscript**. ("For many encoding projects, [...] the problem of overlapping hierarchies is a serious drawback" says Pierazzo (2015, pg.120).)

Only TEI dedicates a whole chapter of its specification to the problem of tree-free structures (TEI-Consortium 2016, sec.20).

3 Text Models and Their Components, As Realised in Different TMFs

As mentioned above, the element types of text models which are well-known to the users and treated by them explicitly are called "component (types)". Informally collecting component types based on manifold practical experiences with different TFMs brings up a list like "paragraph", "section", "title", "table-of-contents", "footnote", "table", "list", etc. These can be seen in the titles of the following subsections, which discuss the component types, their possible mutual relations and their general properties in the TMFs. We try to separate the different component types cleanly, but in some cases overlaps and blurred borders are unavoidable,

When the instances of a component type come in different flavours or variants, these are called *kinds* while the collection of types discussed in this article is fixed, the kinds can vary, even with each document. For each component type the *appropriate mathematical devices* are identified for a further, more detailed modelling, which is omitted in this introductory paper. Whenever no easily applicable exact mathematical method can be found for an empirically given *technical* text format (meta-model), this indicates a design flaw.

3.1 Explicit Hierarchical Structure and Sections

For the purpose of the next subsections, a text can be seen as a flow of characters and mark-up, both abstracted from their concrete appearances (Huitfeldt 1994). Line breaks, page breaks and extra vertical space are possibly included in the mark-up, and additional vertical separators ("asterism", "volute") can be modelled as special characters or as mark-up.

When reading a traditional paper rendering in textual order, a two-stage parsing process is executed by the perceiving human mind: first the mark-up separates the text into *physical paragraphs*. These are sequences of contiguous characters (plus maybe mark-up), separated by changes of appearance, line breaks or/and vertical distances.

In a second step, some of these physical paragraphs are identified as *section title paragraphs*, each of a particular *section title kind*. This can be caused by mark-up, or by the character data containing some key words, or by a combination of both. All text following this section title is perceived as a *section*. Translated literally from Latin, this means that such a title "cuts" the text into parts. The section title kind is transferred to the section as its *section kind*. The constructed mental model implies that such a section "contains" all the following text up to the next section title of the same kind, which ends this section and starts the next. So the text is regarded a sequence of sections.

A *most simple section parsing process (MSSPP)* provisionally describes the operation of a reader's mind when constructing the mental model of the text when consuming it in textual order. MSSPP supports nested application of this mechanisms and works as follows, in the way of LL(1) parsing:

All recognised section kinds are stored on a mental LIFO stack (= a "Last In First Out" storage). As soon as a section title is reached with the same kind as one already somewhere on the stack, all sections up to and including this stack level are regarded closed and the stack is shortened accordingly. Then the new section kind is pushed on the stack and the sectioning process applied recursively. Thus an *explicit hierarchical structure* is layed over the text body; each section title and its section appear on a particular level of this hierarchy. The result is a mental model of the text as an *hierarchically organised tree structure*.

Possibly the section text contains physical paragraphs between the section title and the title of the first subsection. These we call *pre-paragraphs*. Due to the traditional reading (="parsing") process, there cannot be any "post-paragraphs" following the last subsection, because only the beginnings, not the endings of these are visibly rendered.

Additionally this hierarchical structure can be made more explicit:

(a) There can be a *naming scheme* which assigns a word like "Part", "Chapter", "Section", "Paragraph" as a *kind name* of a section kind, and which will be used when referring to some text position, see Section 3.9.

(b) There can be a *numbering scheme*. This assigns an individual *sequence number* to all subsections of a particular section. The concatenation of the sequence numbers of all containing sections, in top-down order, yields a *numeric coordinate* for each section.

The older fashion is to use different number systems for different section kinds, yielding coordinates like "A I 1 a α"; the modern approach is just to use decimal numbers, yielding "1.1.1.1.1". Both have their merits.

(The fact that this nesting of sections and subsections obviously, even necessarily forms a *tree-shaped graph* misled the inventors of SGML/XML et alii, to model *all* *levels* of a text as a tree, which is of course in no way adequate and causes severe problems nowadays when trying to specify and implement sensible processing algorithms !-)

The section title paragraph contains a *title text*, which serves as a name and/or as a description of the section's contents. The title text may be the *empty string*.

Typically, in the text contents of the title paragraph, the kind name and the section's sequence number or the complete numeric coordinate *may* precede the title, together with some constant decoration or separation stuff. In case of an empty title text, one of these *must* appear. Different combinations are possible, like "1.1.1.1 Introduction" or "Subsection 1: Introduction" or "1 – Introduction".

Normally these rendering rules are constant throughout the whole text for one particular section kind/section title kind.

A common practice is that the sequence number (or the complete coordinate) appears only for all sections higher than a particular hierarchical level.

(Again, substantiality is critical, and basically three strategies are applicable: (a) The fundamental rules for the rendering of the section kinds are substantial and encoded, and only the title text (as the only varying data) is encoded explicitly with each section; (b) additionally the number is made explicit, but only as an abstract mathematical value, its rendering is defined by the data from (a); (c) the complete concrete contents of each title paragraph is considered substantial and is encoded.)

The following text is an example for the parsing of a three level sectioning with minimal means, namely line feeds as only mark-up, just to constitute the physical paragraphs, and explicit numbering and naming for identifying the paragraph kind and section title kind:

```
Chapter 1

1:

First Letter

Dear Friend!
Here in Transsylvania ...
```

Normally the sequence numbers of sections of all kinds are positive, start with one(1) and grow upward by one(1). But this may be different for every section kind individually: While counting down or jumping non-monotonously is quite uncommon, a zero-based counting scheme may be sensible. Furthermore, when modeling legacy texts, some sections may have been lost and gaps in the numbering are substantial.

Normally the counter for all sections of kind sK is reset when the immediately containing section is closed. But it is also possible to bind this reset to the closing of a particular kind sR somewhere higher in the hierarchy, or to suppress it completely. This we call *numbering reset scope*. (E.g. it is common that "Chapters" are not reset when a new "Part" starts.)

In most texts the intended *explicit hierarchy* adheres to some of the following properties:

- eh-1. No section contains directly sections of its own kind.
- eh-2. All sections of a particular kind sk1 contain directly only sections of some ($n \geq 0$) particular kinds sk2, sk3, ..., a proper subset of all section kinds.
- eh-3. All sections of a particular kind sk1 contain directly maximally sections of maximally one ($n \in \{0,1\}$) particular kind sk2.
- eh-4. All sections of a particular kind sk2 are contained directly only in sections of one particular kind sk1.
- eh-5. All sections of kind sk1 contain directly at least one section of kind sk2.
- eh-6. All sections of kind sk1 contain directly zero(0) or the same number $n \neq 0$ of sections of kind sk2.
- eh-7. When a section of kind skX contains directly or indirectly a section of kind skY, then no section of kind skY contains directly or indirectly one of kind skX.
  (MSSPP violates this for an input sequence of title kinds like T1–T2–T3–T1–T3–T2.)
- eh-8. Either none or all sections of a particular kind sk1 contain pre-paragraphs.

eh-9. Either none or all sections with subsections contain pre-paragraphs.

All texts fulfil property eh-1, by definition.

Algorithm MSSPP ensures eh-1. But it is too restricted: every individual section can contain only sections of one particular kind. Each section title kind starts either a "sibling" or an "aunt" of the current section, if it is already contained somewhere in the current state of the stack. Otherwise it starts a "child" on a further stacking level. The case that a new section title starts a "sibling" with a *different* kind can only be recognised by additional and explicit rules, mentioning particular title kinds.

Most contemporary scientific papers fulfil eh-3 and eh-4 for all section kinds, by intention. This implies eh-1, eh-2 and eh-7 for all section kinds and establishes a one-to-one map between section kinds and levels of hierarchy. The structure of the hierarchy may vary only by partly omitting the lower levels. Fulfilling additionally eh-8 is often estimated good style.

Adding eh-5 for all kinds but that on the leaf position makes that the hierarchy has the same depth at every leaf. This is a property often emerging by chance.

The combination of eh-3 and eh-4 may be weakened by the possibility to "cut out" one particular level in the middle of the hierarchy. This technique can be found together with the heterogeneous number coordinates in older books, e.g. that Part A consists directly of chapters A.1, A.2, but part B has an intermediate grouping level B.i.1, B.i.2, ..., B.ii.1, B.ii.2, etc. For this, MSSPP suffices.

Adding eh-6 to eh-3 and eh-4 yields the (in-)famous dialectical structure of Hegel's "Enzyklopädie der philosophischen Wissenschaften".

Many contemporary scientific text books fulfil eh-3 and eh-4 for all section kinds, with one very specific and limited exception, namely that each section of kind "Chapter" begins e.g. with a section of kind "Survey" and ends with a section of kind "Exercises", both not included in the numbering scheme which covers the "normal" subsections in between. So eh-3 is replaced by eh-2 for kind "Chapter", and enhanced by eh-5 and eh-6 w.r.t. "Chapter" vs. "Survey" or "Exercises". (This is a phenomenon of heterogeneity, similar to the existence of pre-paragraphs, and could be modelled more adequate by a grammar based approach, like the implicit hierarchies discussed in Section 3.2.)

Usually, the explicit hierarchical structure is represented in a *Table of Contents (ToC).* This is a formatted piece of text in which every section of a particular range of hierarchy levels is represented by one or more text lines. Like in the title paragraph, different aspects of each sections can appear in different forms:

(a) the numbering, either (a1) only the sequence number or (a2) the complete coordinate;
(b) the section kind, either (b1) explicitly by the kind name, or (b2) implicitly by optical appearance, or (b3) by both;
(c) the title text;
(d) some navigation means to the start of the section's rendering, mostly a page number.

The selection and way of presentation of these pieces of informations can be defined individually for every section kind. In any case, the hierarchical structure must always be visible, at least by some mark-up like font size, indentation or leader lines.

(It is again subject to definition whether all these arrangements are substantial or accidental, as discussed in Section 1.3.)

Instead of the title text, each section may be assigned a *ToCTitle* to appear in the ToC, and additionally a *running title* to be printed in the page header or footer in some paper rendering. These alternatives are sometimes required for optical reasons, when the title text is too long.

There may be more than one Table Of Contents, e.g. (a) one survey omitting lower levels of the hierarchy, (b) one overall and detailed, (c) one for each part, placed at its beginning, (d) one for all appendices, before the first, etc. Each of these tables shows a different subset of sections, filtered by horizontal and/or vertical position.

Manuscript and **typescript** first require linefeeds to constitute a physical paragraph, and then mark-up (underlining or upper case only, etc.) and/or special text contents (leading numbers and keywords) for clarifying that the paragraph is a section title of some particular kind.
ToCs must be constructed explicitly, thus are error-prone.

LaTeX comes with a predefined fixed hierarchy of section kinds, per "document class" (Lamport 1986). The numbering is one-based, contiguously increasing and rendered as decimal numbers. Numbering is printed for all sections above a particular level, defined by the counter `secnumdepth`.
Arbitrary jumps in the numbering can be achieved by manipulating the corresponding counters manually. One alternative short title can be supplied, for use by both, ToC and as running title in page headings.
The ToC lists all sections above paragraph level, with title text, numeric coordinate and page number. For higher level sections the kind name is printed before the number, as in "Chapter 12", but only in the section title, not in the ToC. The wording respects the main human language of the document.
When only the predefined section types are used, properties eh-3 and eh-4 are fulfilled for all these, implying eh-1, eh-2 and eh-7. But additional section types may be defined by deriving from a generic "`section*", "subsection*`", etc. environment, which starts a section not interfering with the numbering scheme of the standard sections. This breaks eh-1 and eh-3, but still ensures eh-4 and eh-7.
(As mentioned above, LaTeX is a Turing complete language and the appearance of everything can be tailored according to the user's needs. So the description above is only about the off-the-shelf standard settings. The same holds for **Lout, HTML, d2d_gp,** etc.)

Lout: The additional "Document Layout Package" provides chapters, sections, automated generation of ToC and page headers, etc. but no free recursion into arbitrary deep nesting, because "the author considers sub-subsections to be poor style". (Kingston 2000a, end of 4.4) Using these library package, a ToC is constructed automatically, employing an auxiliary data base.

DocBook has a predefined fixed hierarchy of section kinds, ensuring properties eh-3 and eh-4, implying eh-1, eh-2, eh-7. (Walsh 2010, sec.3.4).
Numbering of sections is always implicit and cannot be specified in the document. A ToC can be included explicitly, but this is not recommended: it should be generated automatically by the presentation/rendering process. (sec.9).

TEI models sections explicitly, by proper nesting of `<div1>, <div2>`, … or of `<div>` elements. The numbering can be given explicitly as attribute value `@n` (TEI-Consortium 2016, sec.4.1.2). The nesting level is implicitly defined by tag nesting. The section kind can be freely given as `@type`, and no rules apply, only recommendations, so none of the properties listed above is ensured.
Initial contents of a section can be a `<head>` element, which specifies the title text, or a sequence of more than one of these, each with a different `@type`. **TEI** and **HTML** are the only frameworks which allow more than two (>2) title texts for the same section.

HTML does not model sections but only the title paragraphs. Recommended usage is `<H1>` elements for the toplevel section titles, `<H2>` for the titles of the next lower level sections, etc.
There is no restriction on titles; different values of `@xml:lang` or `@class` make sequences of more than one title element sensible. **TEI** and **HTML** are the only frameworks which allow more than two (>2) title texts for the same section. There are no notions of ToC, kind name or numbering scheme. (But the latter is modelled explicitly in CSS 2.0.)

OD-T also models title paragraphs not sections. The standard explicitly defines sections to extend from one title paragraph "to the next heading at the same or higher level" (Durusau and Brauer 2011, sec.5.1.2). This seems similar to MSSPP. But indeed it is quite different, because here "level" and "kind" (taking `@style` as such), are both explicit. So different kinds can appear on the same nesting level (=as siblings) without problems, and same kinds as parent and child. Indeed, *none of the properties from above* is guaranteed.

OD-T foresees different kinds of indexes. These include a generic one for user extensions, and a particular ToC kind. Into this section titles are collected automatically, combined with explicit entries by the user. (sec.8)

d2d_gp realises sections by nesting of `<h1>, <h2>`, ... or of `<Hrec>` elements, similar to TEI. ToC construction is implicit, Two(2) alternative short titles are supported, to appear in references and in the ToC.
Numbering is implicit and appears only in a back-end format. In the current default rendering rules, number formats are fixed to decimal. There may be one single appendix, the Chapters contained therein are numbered with uppercase alphabetic characters. Section kinds are implicit, i.e. given by the nesting level, trivially ensuring properties eh-3 and eh-4, implying eh-1, eh-2, eh-7.

3.2 Implicit Structure and Paragraphs

The physical paragraphs which are not title paragraphs make up the "contents" of the sections. They are simply called *paragraphs* in the following. A sequence of paragraphs is either the only contents of a section at leaf position, or forms the pre-paragraphs preceding its first subsection.

Each paragraph has a *(paragraph) kind*, which is recognisable by mark-up or by some leading character data or by both. Many texts will only have one single kind of paragraphs, the contents of which is flow text, making up the contents of the section. Frequently found other kinds of paragraphs are: Motto, Quotation, Survey, Comprehension, Deviation, Source Code Example, Screenshot, Music Notation.

There is a further level of hierarchy below paragraph, namely *line display* of embedded mathematical/chemical formulas, or embedded music notation, etc., framed by line breaks. Sometimes such a combination is not clearly distinguishable from a sequence of separate paragraphs. But it is clearly no such sequence when the line display is part of a sentence, e.g. when first demonstrating

$$1+2=3$$

and afterwards continuing the speech.

This distinction can be relevant because paragraphs must possibly be also numbered for text navigation, implicitly, see Section 3.9. Nevertheless also explicit numbering can be applied to the lowest levels of hierarchy, as it is usual e.g. in the bible or in printed laws. Then the discussion of the numbering scheme for sections from Section 3.1 applies accordingly.

Also Definitions, Lemmata, Theorems, Theses, etc., esp. in mathematical texts, can be considered special kinds of paragraphs.

In general, the notion of paragraph is the fundamental organisational unit for line break calculation. In most MSF, the paragraph notion is *not directly compositional*, but only indirectly: Paragraphs may contain tables, which contain paragraphs, etc.

The sequence of paragraphs make up the contents of a section, and by the sequence of their kinds they constitute an *implicit hierarchy*, which is a kind of "down-side prolongation" of the explicit hierarchy of sections. This hierarchy is defined by rules which refer to the possible sequences of paragraph kinds, and which are most naturally expressed by *grammar rules*.

E.g., in a mathematical text a paragraph of kind "Theorem" will be presented with special graphic appearance, numbering, etc. It is common habit that immediately afterwards a sequence of paragraphs follows which contain a proof of that theorem, the last of which terminates with a "q.e.d." symbol. Alternatively it may follow a single paragraph which explains why a proof is omitted.

Similar, after a paragraph of kind "Definition" there may follow some paragraphs of kind "Example". Furthermore there are "Normal" paragraphs without any special role. These rules are most adequately modelled by a context free grammar over paragraph kinds, as shown in **Figure 2**.

TheoremAndProof ::= Theorem, (Proof | NoProof).
Proof ::= Paragraph^{+}, Paragraph_qed.
NoProof ::= Paragraph.
DefinitionAndExample ::= Definition, (Example*).
Example ::= Paragraph.
Normal ::= Paragraph.
Section ::= (Normal | DefinitionAndExample | TheoremAndProof)$^{+}$.

Figure 2: A Possible Grammar Over Paragraph Kinds.

The nesting of recognised non-terminals, the “parse tree”, constitutes above-mentioned implicit hierarchy, extending most naturally the explicit hierarchy of sections. (Please note that these rules are valid, but mostly not “reified”, ie. they do not appear explictly in the DTD grammars which come with the commonly used TMF implementations.)

Again, the borders are not hard: The example with the “Exercises” section at the end of each “Chapter” can also be modelled by dedicated paragraph kinds, in the realm of the implicit structure. Perhaps even more naturally.

As to a section kind, an explicit *paragraph kind name* and a numbering scheme may be assigned to a particular paragraph kind. The kind name may appear printed in the rendering or be used only for constructing references.

The numbering has a numbering reset scope, as defined above: it can restart at each section, but in most cases some higher level section kind triggers the counter reset. So a complete numeric coordinate can be constructed by appending the individual number to the coordinate of the section of that level. The numbers can be calculated for paragraphs of all kinds separately, or for several kinds together. Most other considerations on numberings from Section 3.1 apply accordingly.

Sometimes inventory lists = *indexes* are included in a text or rendering, referring to all paragraphs of selected kinds like a Table of Contents refers to sections. The possible variants discussed above apply accordingly.

3.2.1 Two-dimensional Rendering of Margin Paragraphs

Up to now a text model is a one-dimensional sequence, containing character data and mark-up. A critical phenomenon are *multi-column paragraph renderings*. This means that different categories of paragraphs are arranged horizontally, and their vertical alignment establishes a semantical relation. (This must not be mixed up with a through-out multi-column rendering of a one-dimensional text, where vertical positions meet *accidentally*.)

Mostly a main text is placed in the inner column of each page, and a smaller outer, marginal column contains additional information, or comprehensions or examples. These all are rendered as full-fledged paragraphs on their own, containing lists and images, etc., but often in smaller fonts. Or the marginal matter are only simple highlighted keywords, change bars, warning signs. These may be set in larger fonts.

In our abstract model, all this can be realised by dedicated paragraph kinds, coming with additional rules. These say (a) that all paragraphs of category cm are related to the nearest preceding paragraph of category cc, (b) cc is rendered normally in the *c*enter of the page layout, and (c) the cm-s are rendered in the *m*arginal space and start at the same vertical position as cc. (These rules may conflict with general rendering rules, and further conflict resolving rules may be necessary.)

(An alternative modelling could put the whole text into one large "Table", but this seems much less adequate for the common situations.)

(This mechanism is very similar to editorial annotations which are logically interspersed into the flow text, but also rendered in the marginal space, as it is common practice e.g. to indicate the page numbers of some reference edition.)

Manuscript and **Typescript** can create paragraph kinds and theorems etc. arbitrarily, by explicit keywords or visual mark-up. Index generation is (naturally) manual and error-prone.
Margin paragraphs are always possible, – famous is e.g. the layout of "Zettels Traum" by Arno Schmidt, a work, in which the geometric arrangement of text becomes substantial.

LaTeX has a mechanism for defining new theorem-like paragraph kinds (Lamport 1986). Counters can be linked to one or more of these kinds, and the numbering reset scope can be set to any of the containing hierarchy levels.
The basic version does not foresee an automatic index generation for Theorems, etc., but this is easily programmable.
Additional library packages allow paragraphs to be rendered on the margin ("`\marginpar`").

Lout has paragraphs as the central organising unit for line breaks, with different strategies selectable per paragraph (Kingston 2013, pg.17). It has a display construct, which seems to break the paragraph notion (pg.33).
It has a predefined collection of special paragraph kinds (theorems, lemmas, corollaries, etc.) (pg.45) and a dedicated construct for margin notes (pg.43). (We could not find out whether these are full-fledged paragraphs.)

DocBook has an extensive but fixed collection of paragraph kinds related to its original purpose (=user documentation for information technology) like `<caution>`,`<important>`,`<warning>` … (Walsh 2010, sec.3.6.3).
Some of them respect whitespace and newline in the source text, like `<screen>` and `<screenshot>` (sec.3.6.4).
Some of the are even more specific for **DocBook**s application realm, like `<procedure>`, `<cmdsynopsis>`, `<funcsynopsis>` and `<classsynopsis>`, which are used to document command lines, as typed to a computer shell, "API" interface calls, etc. (sec.3.6.11).
Mathematical formulas can appear as dedicated paragraphs or "displays" by `<equation>`, and contain **MathML** source (sec.3.7.4) (W3C 2014). (They also can appear inline, see Section 3.6.)

TEI has only one single paragraph kind, tagged as `<p>`. There is no ubiquitous attribute like `@class` in **HTML**, so new flavours of paragraphs must be added explicitly by some customisation as new element tags.

HTML does support different paragraph kinds and different rendering by the `@class` attribute together with CSS or XSLT style sheets. Numbering (of theorems etc.) can be automated using the counters introduced by CSS2. Margin paragraphs are out of the scope of pure **HTML**, but can be realised by the explicit positioning means of CSS3 and ECMA script.

OD-T specification says explicitly "paragraph […] is the basic unit of text in an OpenDocument file." (Durusau and Brauer 2011, sec.5.1.3) Different flavours of paragraphs can be defined defined using the "`<style>`" mechanism.

d2d_gp allows a "paragraph kind" for each paragraph. (Its role is similar to a "CSS class" and indeed translated into such when rendering into XHTML 1.0.) The current standard back-end translations do not support Theorems with numbering, etc. But the "floatings" can be abused for this purpose, as long as they do not really float, since they provide automated numbering and index generation.

3.3 Two-Dimensional Constructs

3.3.1 Lists

So far text is a *one dimensional* sequence of character and mark-up. The mental model created by its reception but can be seen as something two-dimensional, having two axes, one for the parent relation and one for the sibling relation.

The next component type for structuring is a *list*, which can be seen as contained in a paragraph, and in turn contains a sequence of *list entries*, which are sequences of paragraphs. A list has a lead-in marker, preceding each list entry. This may be (a) one fixed symbol, like "+" "–" or "•", indicating only the level in case of nested lists, or (b) a sequence number of the list entries, from one selected numbering system, or (c) character data, individually for each list entry. In case (b) the same variants are possible as discussed with section numbering, see Section 3.1, esp. w.r.t gaps and numeric systems. Case (c) can also be modelled as a variant of tables, see below. The lead-in text is doubtlessly substantial, in contrast to the cases (a) and (b).

A list can be considered two-dimensional because it is constituted by a "horizontal indentation" and a "vertical text flow". But it can also be modelled as a second prolongation of the overall hierarchical structure of the whole text, here becoming *explicit* again.

A central difference between the TMFs is, whether lists (and the other component types of this section) are (a) contained in paragraphs, or (b) siblings to paragraphs, on the same level. Due to the central role paragraphs play, see the dedicated Section 3.2, we prefer (a), because (b) would require to double instantiate all the grammar based logic described above and is farther away from compositionality, see Section 2.6. The property (a) we call **ListSubP**.

Manuscript and **Typescript**: arbitrarily deep nested lists can be constructed, normally by indentation and one dedicated prefix for each level.

LaTeX and **Lout** support arbitrarily deep nested lists. The lead-in symbol can be a number from different numbering systems, or a fixed symbol, or some user-defined text, changing with each item. The value for starting or continuing numbering can be overwritten. Lists are fully compositional: list items can include arbitrary text, including lists, paragraphs, tables, graphics, etc.

DocBook supports a collection of lists which are specialised for particular purposes, violating the SoC principle (Walsh 2010, sec.3.6.3).

TEI is one of the few frameworks which allows title texts for lists. (This could be regarded as a slight violation of SoC.) It allows explicit numbering of list items or label text (TEI-Consortium 2016, sec.3.7).
It is the only TMF foreseeing an *explicitly selectable inline format* for lists which (a) enumerate different topics, but (b) are part of the same sentence and thus (c) typeset in one line. (So these kinds of lists are visually "one-dimensional"!) This is a step towards *linguistic analytic mark-up*, which can go even further and finer, see Section 3.4.

HTML Supports lists with numberings, with constant lead-in symbols or with varying lead-ins (`<ol>`, `<ul>` and `<dl>`). Details of numbering and optical appearance are delegated to CSS 2.

OD-T Different flavours of list can be defined using the "`<style>`" mechanism. **OD-T** also allows title texts for lists (Durusau and Brauer 2011, sec.5.3.3).

d2d_gp The basic default model has an utmost simple list model with different kinds of lead-in symbols or number formats. (Due to the principle of compositional and re-use, this model is meant to be replaced by a more elaborate one, according to the user's needs.)

XML in general does know nothing about paragraphs, because these live on the level of a concrete XML instance.

3.3.2 Tables

Another two-dimensional component type are *tables*. Their construction and definition are non-trivial, as the complicated historic process of the "**CALS** table model" shows (Bingham 2000).

For a simple model, it is sufficient to encode each *table entry* by its contents plus a pair of coordinates which gives the starting row and column. Each entry extends to the next higher entry's coordinates, as shown in the left diagram in **Figure 3**. The identity of the text model is not defined by the sequential order of these table entries as represented in the source text, but by their effective coordinates.

In contrast, all bespoken TMFs except TEI do map the sequential order of the sources of the cells to the sequential order of the cells in the model: They follow source model coupling (SrcMC), as defined above in Section 2.3. This is (a) of course very convenient for simple cases, but (b) it is not a necessity (coordinates could be given explicitly) and (c) it restricts the expressiveness for more complex shapes of cell unions substantially: the cell structure in **Figure 3(A)** we call "L-shaped" = "*table-L*". It cannot be described by pure table-SrcMC, – it requires explicit start coordinates. The cell structure (B) we call *isolated table cells* or "*table-I*". It needs even more explicit input parameters, namely additional explicit *end* coordinates.

The rendering of the model becomes especially critical in case of speech synthesis, where the text has to be rendered line by line or column by column or somehow navigable. Whether layout information (e.g. relative column width values) is substantial or accidental is once more a critical question.

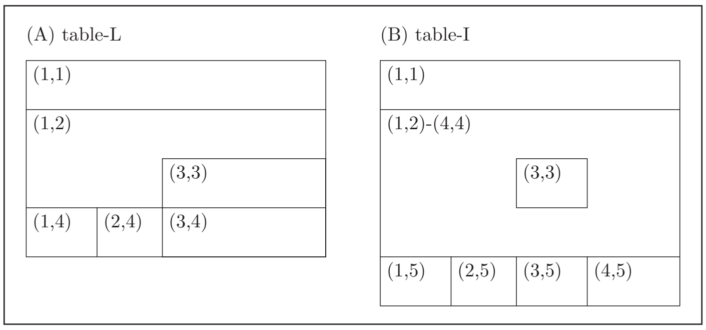

Figure 3: Table Cells De-Coupled From the Source Text Order.

On model level, tables can also be constructed with *more than two dimensions* (>2). Most simple example it the *tabbed stack of input forms*, ubiquitous in many Graphical User Interfaces (GUIs). These have "two-and-a-half" dimensions, because the positions on the third axis (= z-axis) are less strongly related then columns and rows.

Furthermore, table entries can be arranged truly spatial, in a grid of cubes, and in dynamic digital rendering this grid can be presented with varying aspect angle and eye position, controlled by the reader. This is hard to render and seldom found on paper. We expect it to become more frequent with the development of Computer-Aided Reading (CAR).

Additionally, there may be a non-local property of tables which leads to an *inter-cell alignment* of text entries in all cells of the same column and different rows, controlled by some alignment character.

With the component types introduced so far, the *principle of compositionality*, see Section 2.6 above, begins to show its fundamental relevance: May lists contain tables? And tables lists? And table entries paragraphs, etc.?

There are more complicated flavours of tables, where the text entry is enriched by graphic elements, e.g. different styles of cell borders may carry some meaning, or free connecting lines between cells may indicate relations. This is on the half way to *free diagrams*.

Manuscript and **Typescript**: arbitrary (two-dimensional) tables can be constructed. In a limited way, two-dimensional pictures of three-dimensional tables can be drawn.

LaTeX has two sophisticated basic table models, for text mode and for math mode, resp. Both follow table-SrcMC (Goossens and Mittelbach 2004, pg.239). In principle, both are fully compositional. Column width can be left for automated calculation, or set explicitly by the user (normally calculated from the text dimensions dynamically).

Multi-column entries are supported, but no arbitrarily shaped multi-row-and-column segments.
The separating insets and borders can be defined in a very flexible way.

Lout has an elaborate table model (realised as an additional package), supporting colspan, rowspan and allowing page breaks within tables. It follows table-SrcMC (Kingston 2013, pg.125).

DocBook imports two(2) different types of table definitions: from **HTML** and from **CALS** (Walsh 2010, sec.3.6.5).
The CALS table model is a historically important and rather elaborate one. (Bingham 2000; Walsh 1999) It supports source-model-DE-coupling, i.e. it allows to define table cells in any source order, it uses table-SrcMC only as its default mode. The CALS model supports one *alignment character* per entry, which refers to other entries in the same column.

TEI has a simple table model, following table-SrcMC. It additionally defines a versatile `@role` attribute for table cells. The documentation explicitly mentions the alternative to integrate an XML declaration of the CALS model (TEI-Consortium 2016, sec.14). (What is called "alignment" in TEI are heavy-weight means on a semantic level and not comparable to a simple alignment character (sec.16.4).)

HTML comes with a very a simple table model as its standard. It is restricted to table-SrcMC. Nowadays rendering information will be added using CSS (W3C HTML Working Group 2011). CSS 3.0 introduces an alignment character by `text-align=","` (W3C HTML Working Group 2011, sec.7.1).

OD-T: The table model is re-used in text as well as in spreadsheets. Basically it is free recursive. The denotation of tables follows table-SrcMC. Different flavours of table can be defined using the "`<style>`" mechanism (Durusau and Brauer 2011, sec.9).

d2d_gp The basic default model has an utmost simple table model. It follows table-SrcMC.
(Due to the principle of compositional and re-use, this model is meant to be replaced by a more elaborate one, according to the user's needs.)

XML in general does know nothing about tables, because these live on the level of a concrete XML instance. But there are standardised XML encoded table models available (like XHMTL tables or CALS) which can be imported and used by any user defined architecture.

3.3.3 Diagrams, Figures and Pictures

Free diagrams combine graphical elements with textual elements. Due to compositionality, the textual parts can contain everything like a flow text paragraph. Beside the graphic components carry substantial information, the textual components can nevertheless still be approximated by modelling it as a table, i.e. arranged in rows and columns.

A possibly substantially relevant new kind of information is that text can appear *rotated*. Already in simple tables a rotation of plus or minus ninety degrees is sensible, but perhaps not substantial. In free diagrams any angle may occur.

When including diagrams and pictures in some graphic file format, it may be a severe issue that text may "appear" therein, readable by the human eye, but not part of the text model. This should be avoided, since automated processing is impossible, and the reader can be irritated by failing search or search-and-replace attempts.

Therefore a better solution is CmpSS, defined in Section 2.4, which integrates the source text of the diagram, which in turn contains the text components with a readable and searchable representation.

Manuscript and **Typescript**: arbitrary diagrams, figures and pictures can be included.

LaTeX has no genuine support for graphics, but a plethora of additional packages exist. There are two groups of additional functionalities: One to insert existing graphics into the rendered text flow, like **postscript** or PDF files, pixel graphics in `png` or `jpeg` format, etc. The main purpose of these functions is to integrate the graphics in the layout process of the different pages, so only their "*bounding box*" is relevant. Some of these functions allow to re-scale the input data, but the result may be insufficient.
The other group are *graphic construction languages*, like `pstricks` and `pgf`, which allow to describe a graphic directly in the same source text with the surrounding text (Compound Source Strategy (CmpSS), see Section 2.4).

Lout has embedded graphic and diagram description languages, thus it follows CmpSS, see Section 2.4. (Kingston 2013) It is very elaborated and versatile for approx. six different realms, but limited to these. Furthermore, like **LaTeX**, it can embed externally given graphic files.

DocBook has no embedded graphic language, but must import graphics in an opaque way (Walsh 2010, sec.3.6.8).

TEI: The element `<graphic>` includes a graphic object from some external file, identified by its URL (TEI-Consortium 2016, sec.3.9). There is an additional `<figure>` element which allows grouping of these insertions (sec.14.4). According to its genuine use cases, there is a second element for grouping, namely `<facsimile>` (sec.11.1).
While the general purpose graphic language SVG is not included per default, but needs explicit customisation, the standard comes with a libraries for specialised graphics, namely "Graphs, Networks and Trees" (sec.19). These all follow the CmpSS.

HTML has from its first days means to embed external graphic objects, in different bit map formats. Contemporary technology supports SVG, which allows to embed graphic source text directly in the document (CmpSS).

OD-T has an integrated drawing language. Their elements can unrestrainedly by mixed with text elements. (Durusau and Brauer 2011, sec.5.1.3, sec.10) The principles of SoC and reusability are heavily violated, e.g. there are more than one kind of "anchor" elements, all doing the same and appearing everywhere. Again, **OD-T** seems a particular bad design.

d2d_gp The standard text model has a simple HTML-like element for graphic inclusion.
But the main strategy for graphic inclusion is CmpSS, and source text for different graphic languages can be embedded easily, as it has been frequently done for e.g. the the LilyPond music engraving compiler. (Lepper 2015)

XML in general relies on CmpSS and allows inclusion of other XML based standards. So "Scalable Vector Graphics (SVG)" (W3C 2011) may be included into the document's substance for defining graphics. Such a graphic can in turn include recursively *without a priori restrictions* all types of element and data as the rest of the document.
The main problem in this context is that the different formalisms for document type declaration do not support free recursive instantiation, so that the intended text format cannot be reflected on the specification level.

3.4 Segments of Character Data, Highlighting

It is frequently necessary to select a contiguous *segment of text character data* and render it in a special way, to indicate some substantial property of this segment. As explained above, there are two fundamentally different technical means: tree-bound and tree-free text segments, see above Section 2.7.

The model components *highlighting/segmentation* correspond directly to "mark-up" in the original sense of the word, which once meant to take a transparent marker pen and to draw lines *freely* in a text, crossing borders of sentences, paragraphs and sections. Also arbitrary overlaps with other mark-up segments are permitted, when using different colours, without the need of proper nesting.

This can only be modelled directly by a TMF which supports tree-free segments, including the additional consistency checks, as discussed above in Section 2.7. With tree-bound segments only, such a mark-up must be split into a sequence of element with a "continuous leaf front", see **Figure 1**.

Seen from the application stand point, character data segments fall into very different use cases, with different granularity and complexity. Most of them are totally happy with tree-bound realisation.

First it may be distinguished whether the mark-up and its contents belong to the same or to different substantial layers:

Declaring a segment as "emphasised" or "very emphasised" can be done by an author, when constructing a first layer of a text, thus as part of the substance of this layer. (As discussed in Section 1.3, it is an open decision whether the optical appearance of this emphasis is substantial or accidental.)

But the emphasis can also be applied to that first layer T1 by an editor, and thus be part of the layer T2, as defined above, forming a multi-layer text, see Section 2.1.

Means for rendering are changing the font family or font variant, from upright to slanted, small caps or bold, different ways of underlining, framing, or even different colours for text or background, etc. In any case the substantiality of the rendering must be discussed. Anyhow, if the semantic definition of these segment types requires their *compositionality* (cf. Section 2.6), then this must be supported also by the chosen optical appearance, e.g. by translating to "underlined", "bold" and "slanted", which are freely compositional.

Segments can also be used to identify the contained text as an identifier of a specific kind, e.g. the name of a human or the title of an opus, but this will in many cases imply additional navigation means and thus be treated according to Section 3.5.

The most complex segmentations naturally come from *linguistics*, because there text T1 itself, as such, is the subject of analysis. The mark-up belongs to a level T2 of analysis, has very fine granularity and is tree-free. But also much less complex segments by an editor, e.g. change marks, will in many cases by tree-free.

But there are (a) other categories of segments which are always tree-bound, like personal names, and (b) applications which are accidentally properly nested, like an emphasis completely contained in an embedded foreign language text. For these, some frameworks which do support tree-free mark-up offer additionally a *simplified implementation*, which is again tree-bound and thus verifiable by the document type definition.

3.4.1 Multi-Lingualism

A frequent kind of segmentation, ranging over many paragraphs or only over a small fragment of a sentence, is used to identify text in a human language other than the main language of the document. This is usual for common terms ex gratia from logic and rhetorics, et cetera. In German type setting style, these are marked by fonts, formerly switching from Fraktur to Antiqua, or in modern settings from upright to italic shape, but not so in English texts.

A different form of multi-lingualism is the inclusion of a *whole paragraph* in a different language. This can most naturally be modelled as a dedicated paragraph kind, see Section 3.2. A translation into the text's main language can follow immediately as a second paragraph. This arrangement can be enforced by grammar rules, as described above. Alternatively, the translation can be given as an annotation, see 3.7.

Both granularities of bi-lingualism have their own problems when the *directions of writing* in both languages differ.

A third format of multi-lingualism is a text with more than one (n>1) main languages, a multi-lingual document, where each paragraph appears as a sequence of paragraphs in the different languages. These may be arranged in sequential order, or graphically in parallel. This is a typical phenomenon e.g. for security advices in international hotels and in public transport. And on the Rosetta Stone.

Two further cases are seldom foreseen in TMFs: text segments (a) in an *unidentified languages*, and those (b) *readable in more than one language*.

Case (a) is quite frequent in archaeology: for years documents written in "Linear B" could be treated as texts, namely copied, printed and graphically analysed, but not assigned to a particular language. (This could be called a *language-free text* !-)

Case (b) can only happen in specially constructed texts like poems, riddles or mystery novels.

In **Manuscript Typescript** highlighting is done by underlining, etc., which is tree-free.

LaTeX provides commands for text layout which change font, size, colour, family, etc. These "physical" parameters can be used to stand for "semantic" mark-up, as described above, by macro programming. There are tree-bound and tree-free versions of all these commands, but programming can be become complicated when these are mixed. The mapping from some arbitrary semantic to the pre-defined physical mark-up must be done explicitly by macro programming. This can be specific for one particular text body, or contained in a "style file", realising the standards of a particular series of publications.
The additional package `babel` supports multi-lingual documents and small scope language switching.

Lout has one dedicated command for changing the documents language per text segment. This will affect the line break rules, which are borrowed from TeX (Kingston 2013, pg.25).

DocBook has the inline element `<foreignphrase>` (Walsh 2010, sec.3.7.3). Its language, that of the top-level `<book>` element, and of all intervening elements which support the `db.common.attributes` is set using the standard `@xml:lang` attribute introduction to element reference.

TEI provides some tree-free segmentations (see Section 2.7 above) by "empty" elements like `<n:sentenceBoundaryStart>` and `<n:sentenceBoundaryEnd>` (TEI-Consortium 2016, sec.20.2). All these are dedicated; regrettably there is no generic one.
It foresees dedicated elements to mark tree-bound character segments, mostly following linguistic categories, like `<foreign>`, `<said>`, `<quote>`, `<mentioned>`, `<bibl>`, `<term>`, `<note>`, and few somehow generic like `<emph>` and `<hi>` (= "highlight"), but no real generic one like `<span>` in **HTML**.
The full range of XPointer is incorporated for defining references (sec.16.2.4). With its `range()` construct, tree free segments can serve as the target of a reference.

HTML allows the mark-up of in-line fragments by the element `<span>` and of block elements by `<div>`. Both are only allowed as tree-shaped, and no combination is possible.
The roles of these segments can be defined freely by the `@class` attribute. For the special case of language switch, every element supports the universal attribute `@xml:lang`. So even so simple things like a hard line break `<br>` or a horizontal ruler `<hr>` exist in every thinkable language.

OD-T has tree-free segments for so-called "index marks". These may overlap, but (rather arbitrarily) may not cross paragraph boundaries. The index marks land in one of the two predefined tables (ToC and register), or in a user defined one (Durusau and Brauer 2011, sec.8.1.2-8.1.9).
The same concept is realised for "change marks", in a tree-free and a point-wise (=empty element) flavour (sec.5.5.7).
The more general concept are so-called "bookmarks" and "reference marks" (sec.6.2). Again, SoC does not happen and every feature is defined more than once.
The most general concept `<text:span>` does regrettably only exist in a tree-bound version. It is intended to be linked to layout information (sec.6.7.7).

d2d_gp supports physical highlighting and basic "semantic" mark-up like "emphasised" and "strongly emphasised". All these are tree-bound. User instantiations frequently define their own, which may by tree-free.

d2d_gp has a dedicated "span" for fragments in foreign languages, a dedicated parameter for every paragraph for switching the natural language, and allows to declare multiple base languages per document.

XML in general supports both, three-free and tree-bound segments. It defines a general purpose attribute `@xml:lang` (one of the very few which live in *XML namespace*) which can be attached to any element the user allows, which is defined to indicate one human language for its contents, and which is taken over by all XML based TMFs.

3.5 Entities, Definitions and References

Texts talk about things, these things have names, these names are employed to identify the things. These seemingly simple facts have been discussed by the earliest known philosophers up to most recent schools in scholarship, linguistics, informatics, brain research, etc., and dozens of very different theories have been elaborated. For the discussion of this article only few simple mathematical properties are needed, which can be derived from any theory and serve as a bridge to the merely technical requirements of the TMFs.

In general, it is necessary that the above-mentioned "names" must have the mathematical property of a *function* , which means un-ambiguity: a "thing we talk about" is a kind of concept as part of some mental model, and we call it entity. Un-ambiguity means that each name refers to only one(1) entity. Due to this, the name is said to be an *identifier* in the strict mathematical sense. In the context of a computing algorithm it is furthermore highly desirable that these identifiers are additionally *injective*, i.e. there is only one(1) such identifier for each entity.

There are two possible conflicts: For a human reader, the identifier should be made of words of a natural language. Basically there can be two strategies, "unified name space" or "separated name spaces". With the first strategy, the natural language word refers to an entity unambiguously. There is only one "Beethoven" and one "C major". With the second strategy, the identifier is a pair of the natural language name plus a *realm indication*. So there can be two different entities called "C major", namely a "key" and a "chord", which are two substantially very different things. And the name "Richard III" can denotate a theatre play, a dramatis persona and a historic person, – even all of them in one single sentence. In these cases, when in the foreground of a text a natural language name is used for several things from different realms, then the real identifier (in the mathematical sense) is indeed a tuple of that name plus a realm indication. The latter is present in the structural middle ground of the text model, but not necessarily shown in a rendering.

The second issue is injectivity in the opposite direction: In the front-end text the natural language names may vary, representing the same identifier. We speak of "Beethoven", "Ludwig van Beethoven", "der junge Ludwig" or "the creator of the late string quartettes". So the *external representation* of an identifier is something different than the identifier as such, and this fact must be respected by storing, retrieving, sorting, comparing, etc.

A recent development are *canonical identifiers for entities*, stored in *authority files*. Paper versions had been developed before 1900, and the electronically accessible implementations during the last decades. Prominent examples are GND for the German language, LCSH in US, NDL in Japan, VIAF for international integration (Bennett, Hengel-Dittrich, et al. 2007). None of the TMFs supports this special kind of identifiers directly. The KBSET project for scholarly editing and annotation is based on **LaTeX** and supports the annotation of each entity reference with a coupled pair of both human-readable identifier (=external representation) and canonical identifier (=model middleground) by automated data base look-up (Kittelmann and Wernhard 2016).

Basically there are different cases for using entities, identifiers and external representations:

(a) First, the entity can be assumed to be well-known. Then only references appear in the text. Computer-Aided Reading may add further navigation paths in the text: From the reference to a particular entity one wants to navigate to the very first, the preceding, the next or the very last reference to the same entity.

(b) In many texts, esp. in scientific texts, there are additionally *formal definitions* of the entities (indexed by the identifiers).

Such a definition can stand (b1) at the *first reference* to the entity in text order. This is the normal case for mathematical theorems, etc.

Or (b2) it can be contained in dedicated lists and sections, e.g. in a *glossary* contained in the appendices.

Or it can (b3) appear at some non-first reference in text order. Then the references before this definition are "forward references". Normally these are only acceptable in "pre-material", e.g. in a "foreword" or an "introduction", to refer to some theorem later in the main text body.

In both cases we want Computer-Aided Reading also to navigate to that definition (which in case b1 coincides with the navigation to the very first reference).

Furthermore we often want a list of the text positions (encoded according to Section 3.9) of all references to a particular entity. These lists typically form an *index*. These references may be attributed additionally by some scalar value indicating "relevance", or by the "kind" of the referring text (from the main flow text or from a figure or from a footnote?).

A very frequent setting is this: most index entries are of "normal relevance"; highly relevant paragraphs are marked by bold font page numbers, this includes the definitions; many references of minor interest are totally omitted: references into figures or diagrams are indicated by using an italic font.

All TMF follow "index-ScrMC", i.e. "source model coupling" as defined above in Section 2.3: An index entry has to stand in the source text at exactly the place it is meant to target at. Only **DocBook** has a de-coupled version: The expression

"`<indexterm zone=`*`id`*`>` *`e`* `</indexterm>`" relates the index entry *e* to the whole contents of the DOM's tree node with the XML identifier *id* and can stand anywhere in the source text (Walsh 2010, sec.9.1.1).

A special instance of (b2) are citations and bibliographies. In case of scientific papers, there are often *two* different "definitions" of the identifiers, namely the entry in the bibliographic list (type (b2)) plus a discussion in a "related work" section (type (b3)). In a larger monography the places of citations can in turn be listed by an index.

The presence or absence of a definition also applies to the overall subject of the text as a whole: A biography on Beethoven will not contain a dedicated "definition" of its topic, – or the whole text can be regarded as such a "definition". Contrarily, a mathematical monography on "lattices" will start with an explicit definition of a "lattice".

Frequently the entities appearing in a text are assigned to certain *entity categories*: e.g. in a text on music history there will appear composers, works, tonal keys, instruments, etc. (These categories live in the text middleground and may fall together with the above mentioned realms in the text foreground.) Often these different categories have specific rules for optical appearance. Entities of the different categories may appear in different indexes, even in more than one.

What is not supported in any of the TMFs are *hierarchical identifiers*: The name "Jan-Nydahl-Schule" is the name and identifier of a pedagogic institution, but its first two components form the identifier of a historic person. Similar with "Richard-Wagner-Festspiele". None of the known TMFs does support this systematically.

In **Manuscript** and **Typescript** indexes are normally omitted. Definitions of type (b1) are easily marked up by underlining the external representation of the subject, by indentation or by short lead-in phrases. This can take the form of dedicated paragraph classes, see Section 3.2, and may include numbering.

LaTeX comes with an accompanying program `mkindex`: some macros write out data to a file which serves as input; `mkindex` sorts and re-arranges the entries and generates a file with TeX macro definitions, which is fed back in a second **LaTeX** run. (Goossens and Mittelbach 2004, pg.647) Arbitrary many index tables of very different kinds can be supported, but this requires dedicated macro programming. The difference between external representation and identifier is easily handled, since the index entry does not contribute to the visible text. The data flow is similar for bibliographic entries, using the external program `bibtex` (pg.683).

Lout has an integrated elaborate index generation facility. Index entry points are not visible, which allows separation of identifier and external representation. (Kingston 2013, pg.62 pp) **Lout** has a predefined glossary format; an external representation in the flow text may be linked to the glossary entry, but only when it is verbatim equal to the identifier (pg.58 pp).
Citations and bibliography are very similar to **LaTeX** (pg.112 pp): the documentation claims that both are derived from the same original design by Leunen (1992). There is an external data base of bibliographic entries, which are referred to by keys and rendered according to the parametrisations in the text model. While **LaTeX** uses an external program, the processing of the data (selection of entries, sorting, rendering of citation keys, rendering of the bibliographic data) is integrated into the **Lout** translation process and can be customised by standard means. A new nice feature is `@RefPrint` which allows to render a single bibliographic entry directly, wherever wanted in the main text flow.

DocBook allows both: to notate indexes, glossaries, etc. manually, in a fixed form, as part of the text model, and the automated construction of these tables. The latter is recommended. For index generation, the text positions to appear in the index can be marked by dedicated elements. Since these do not print themselves, the independence of identifier and external representation is easily ensured.

Index entry levels are fixed to dedicated elements `<primary>`, `<secondary>`, `<tertiary>`, thus violating compositionality. Tree-free segments as targets are first class residents, but hard to type since they need XML IDs and abuse the class attribute. More than one index is not supported (Walsh 2010, sec.9.1.1). A very nice and unique feature is the `@zone` attribute, which allows an index entry defined to cover the whole contents of any structural element (chapter, section), as long as this carries an `@XML id` attribute (sec.3.7.3).
A unique feature is that bibliographic lists can be given in one of two(2) forms: "raw" means as data base entries, which will be formatted by the processor. This resembles the treatment of bibliographic data in **LaTeX** and **Lout**, withstanding that here the text model does not contain a single output controlling parameter. The "cooked" form allows to write down for each bibliographic entry the linear sequence of entry components (=field values), still marked up with their semantics, but interspersed with all necessary punctuation. So rendering is restricted to changing fonts and weights, etc., but the finally rendered data is defined as part of the text model (sec.9.3).
Citations into this bibliography are inserted into the main text flow by `<citation>` (Walsh 2010, sec.3.7.2).
`<citebiblioid>` allows to refer to publications *directly* via "doi", "isbn", "issn", "uri", etc., which is a unique and nice feature in many use cases.

TEI has a huge system of element types and attributes for personal names, dates and places (TEI-Consortium 2016, sec.13). The additional "realm indication", required for injectivity, is thus realised implicitly by the XML element type. This is convenient, but not extensible. Esp. unique is the mark-up of uncertain time ranges with lower and upper limits as in

```
<date notBefore="1930" notAfter="1935">Early in the
1930s</date>...
```

(sec.21.2).
A generic XML element for entity references is `<term>`; an explaining glossary entry can be added de-coupled from source text order by referring `@xml:id`

(sec.3.3.4).
Automatic index generation is elaborate; multiple indexes are supported (sec.3.8.2.2). Further there is a module collecting element definitions for "all kinds of [...] dictionaries, glossaries, and similar documents" (sec.9).
According to its genuine use case, there is extensive support for a "critical apparatus", including automated collection of "variants and witnesses", etc., which act like a specialised form of index generation (sec.12).
TEI supports the modelling of *abbreviations* and their *expansions*.

HTML does only support the technical infra-structure for links and anchors. Together with the mark-up (see Section 3.4) and the `@class` attribute, any semantic modelling device must/can be constructed.

OD-T has a built-in mechanism for index register generation, one standard and arbitrary many user-defined (Durusau and Brauer 2011, sec.8). See Section 3.4, where the mark-up of the target character segment is described.

d2d_gp allows the user to define its own entities. Their input side front-end representation is based on character parsers, which allow convenient and human readable notation "in the flow of authoring" for complex data structures.

XML in general does know nothing about concrete entities, because these live on the level of a concrete XML instance. The "`<!ENTITIY..>` mechanism" of XML seems hardly useful for modelling, but possibly as back-end target format.

3.6 Formalised Contents, Icons

The contents of paragraphs are "flow text". Flow text normally contains written representation of sentences in some human language. In most cases this language is the same for the whole text.

Additionally, in these sentences entity references may be embedded which are *more formalised* than the human language's words, e.g. a text on chemistry talks about H_2O or a text on music about $C^{\overset{9b}{7}}$.

These *formulas (from different disciplines)* differ from normal text w.r.t. line and page breaking, graphic layout (see the "baseline skip" between the preceding lines), etc. Possibly they are two-dimensional. Additionally, a formula may serve as an identifier, with all the issues discussed in Section 3.5.

For longer and more complicated formulas, a rendering as a line display, (i.e. framed by line breaks) may become sensible, see Section 3.2. Again the border of substantiality may be critical.

A similar kind of embedding is to treat *icons* or small graphic pictures like words, embedding them into the flow of a sentence. In the last years this became common with "*emoticons*", but there are very early examples like *alchemical symbols*, which can be treated as an *domain specific expansion of the alphabet*, with each character standing for a whole word. Nowadays this can be mapped to "user space" in the spare unicode planes, if it is not already standardised somewhere.

Manuscript: Arbitrary icons and formulas can be embedded.

Typescript: Up to 1980 it was usual to leave gaps when typing a text and insert complex formulas afterwards by handwriting.

LaTeX and **Lout** support two modes for mathematical formulas: either integrated in a line of flow text, or separately, limited by line feeds. Based on these, on the general "physical" text rendering control and on additional graphic functions, the rendering of "formal entities" can be realised arbitrarily, by dedicated macro programming.

DocBook has a whole zoo of in-line elements, more than thirty(30), specific to its main application, namely documentation of computer software: `<prompt>`, `<literal>`, `<computeroutput>`, `<userinput>`, `<replaceable>`, `<accel>`, `<guibutton>`, `<guiicon>`, `<guilabel>`, `<keycap>`, `<keycode>`, `<classname>`, `<constant>`, `<errorcode>`, `<filename>`, etc. (Walsh 2010, sec.3.7) (A little bit more SoC would have made this zoo more tidy!-)

Esp. interesting are `<tag>` and `<markup>`, which render to the visual appearance of SGML/XML source text structures, in the sense of "reflection".
Mathematical formulas can be inserted inline or as a paragraph, and are constructed using MathML (sec.3.7.4) (W3C 2014).

TEI has elaborate support for defining and integrating glyphs (TEI-Consortium 2016, sec.5).
For music notation, mathematical formulas, chemical formulas, etc., third party formats may be integrated by the elements `<formula>` and `<notatedMusic>`. These are treated in a transparent way by TEI, i.e. no inner structure is defined and checked, but this is left to some external language definition, named by an attribute `@notation`. The source text for this external processing may be given by the XML contents of the element (following CmpSS), or by a `<ptr>` element pointing} to some external resource.
In this way, $\TeX$, mathML and openMath can be used for mathematical formulas, and LilyPond, MEI, MusicXML, musixTex, etc. can be used for music notation.

HTML does only support the technical infra-structure for links and anchors. Together with the mark-up (see Section 3.4), the inclusion of images by `<img>` and the `@class` attribute, any semantic modelling device must/can be constructed.

OD-T supports to embed into the flow of text characters (a) graphics defined by the internal graphic languages, (b) graphics from external files, and (c) mathematical formulas.
(For (a) see the contents list of `<text:p>` in Durusau and Brauer (2011, sec.5.1.3); (b) goes via `<draw:image>`; (c) is possible via the nesting `<draw:g><draw:object><math:math>` !-)
Using these, and the character segment mechanisms described in Section 3.4, special text entities can be constructed. They are not foreseen as first-class residents.

d2d_gp allows to embed dedicated modules for formula languages, based its character parser mechanism; `MathML` is supported by the standard text model and its XHTML rendering (W3C 2014).

XML in general does know nothing about formalised contents, because these live on the level of a concrete XML instance.
But nowadays there have evolved standardised and well-proven XML based definitions of formalised contents in nearly every realm of science and technology, from agriculture to theology, from ontology to toaster control, which can (and should) be re-used and imported.

3.7 Annotations, Footnotes, Apparatus

In printed and edited media, *Annotations* are normally rendered by decorating a word or a sentence in the flow text. This decoration establishes the connection to one or more paragraphs of commentary text which is rendered elsewhere and semantically stands beside the flow of reading of the main text. We speak of *annotation mark* and *annotation text*.

The only TMF which supports a graphic indication of the relation between a note and its point of reference (by a straight line, or sim.) is **DocBook** with its "`<calloutlist>`", intended for source code explanations.

(In **manuscript** and **typescript** there are also annotations of a very different kind: connected to the referred text by free line drawings or inter-linear handwriting, not meant as a separate comment but to be inserted into the flow of text. These annotations reflect the creation process and have been resolved by the editing process before printing. While being an important subject of research, see Section 4.3 below, they are of non-digital origin and therefore not discussed in this article. When creating a text with a TMF from scratch, instead source text level comments can be used, see Section 4.4 below.)

The combination of mark types and rendering positions constitute an *annotation kind*. The most frequent type of annotation marks are superscripted Arabic numbers;

a numbering reset scope must be defined, as for any sequence number. But also lower case Latin characters and dedicated symbols can be used.

On the input side, nearly all TMFs follow SrcMC: The input syntax must appear at exactly the place where the annotation shall appear in the flow of text.

On the output side, typical places where to render the annotation text are (a) the bottom of the current page (the annotation kind is called "footnotes"), (b) the end of the current chapter ("endnotes"), (c) a dedicated section of the appendices ("apparatus" or "annotations" in a narrow sense). When commenting program source text etc., also (d) a side by side rendering of commented and commenting text is frequent, the relation between both given graphically, by connecting lines (see the "`<calloutlist>`" in **DocBook**).

It is common practice that annotation texts of one particular kind may contain annotation marks of a different kind. The resulting graph of applicability between all annotation kinds must be *free of cycles.* E.g., the main text bodies in the "Marx-Engels Gesamtausgabe" have a lot of footnotes by the original authors, marked by numbers, and the main text and these footnote texts can carry annotations into the appendix, by numbers in parentheses. Such an annotation text by the editors can grow to an essay on its own and again carry footnotes. So there are *three (3) levels of annotation kinds*, with application rules not allowing cycles.

Manuscript and **Typescript**: it is not easy to place longer footnote text, but in principle arbitrary strategies for applying notes are possible.

LaTeX and **Lout** have a simple and standard `footnote` mechanism off-the-shelf. There are functions or packages for end notes and chapter-wise registers, etc. More complicated annotations can be realised by macro programming. **LaTeX** has ScrMC as a default, but also a de-coupled variant (Goossens and Mittelbach 2004, pg.109), see the example above in Section 2.3. **Lout** supports only SrcMC (Kingston 2013, pg.40).

DocBook supports `<footnote>` only with SrcMC. A valuable feature allows multiple references to the same footnote text (Walsh 2010, sec.3.7.1).
A structure special for the purpose of program code documentation or screen shot explanation is the `<calloutlist>`, which links text blocks to different positions in the graphic object. It has a de-coupled source format. (sec.3.6.2).

TEI has a versatile `<note>` element, which can model end notes, footnotes, and man other kinds of annotations. Both ScrMC and de-coupled versions are supported. (TEI-Consortium 2016, sec.3.8.1).

HTML does only support the technical infra-structure for links and anchors. Together with the mark-up (see Section 3.4) and the `@class` attribute, any more complex structuring device must/can be constructed.

OD-T has an elaborate note mechanism based on the element `<text:note>`, which can appear ubiquitously in text elements. Different kinds can be defined via `<text:notes-configuration>` and `@text:notes-class`; all follow SrcMC (Durusau and Brauer 2011, sec.6.3).

d2d_gp the standard text model and its back-end rendering rules support one level of non-recursive footnotes; both SrcMC and de-coupled source are supported. (More complex architectures can be defined and plugged in.)

XML in general does know nothing about footnotes and notes, because these live on the level of a concrete XML instance. It provides the general `@xml:id` attribute mechanism, which can (and must) be used to link note marks to note bodies.

3.8 Floats and Figures

Floats are components which are related to the overall flow of the text more loosely than the other components. They are (normally) referred to more than once, and always explicitly by referring to their numeric coordinate. As with theorems, the numbering scheme may combine a sequence number with the coordinate of some containing section, and must define the numbering reset scope.

The concrete position of a float in the (one-dimensional) source text does not contribute to its substance; floats are a good example for explicit source/model DE-coupling.

In all renderings, the position of the float w.r.t. the references in the main body text is determined by ergonomic considerations. In paper realisation, the rendering of a float should appear near the place of the first reference. In screen realisation, it may be sensible to show them in a separate window.

Each float is assigned a *float kind*, and a numbering scheme covers one or more selected kinds, as explained above in Section 3.2 for paragraph kinds.

Each float must have a *title text* called *caption* which describes its contents. A table of all floats of a particular kind is normally found at the beginning or end of the rendering, and gives for each its numeric coordinate, its caption and some navigation means. As with section titles, a shorter title text may be specified for these tables. These tables correspond to the ToCs.

Manuscript and **Typescript**: in this medium, nothing can float. But anything can be inserted at any fixed place. If the text model is meant to be translated into print, then (a) the fact that this object will be allowed to float, and (b) constraints on its positioning must be given explicitly, by some *meta information*.

LaTeX and **Lout** support floating objects and the generation of index tables, based on their captions, per category. The position of these tables and their attributes are part of the text model. (New categories can be added only by macro programming.) The automated positioning of the floats is done heuristically, – quite frequent are situations where the outcome is not satisfying and needs user intervention (Lamport 1986).
Lout allows figures with "fixed positions" to be overtaken by "floats", so the sequential order and assigned numbers in source and outcome may differ (Kingston 2013, pg.48).

DocBook supports `<figure>`, `<example>` and `<table>` as floating objects, with caption, numbering, etc. Whether the object really "floats" in the rendered output, and whether it is listed in an initial "list of figures", depends on the rendering process.

The construct `<table>` confuses a category of floating objects and the internal table structure. The documentation is unclear how to define "non-floating" tables, i.e. tabular structures as part of the main text run. This is a severe violation of SoC (Walsh 2010 sec.3.6.5).

All floats must be given an `@xml:id` attribute to be addressable by references from the main text, see Section 3.9.

TEI has a generic `<floatingText>` construct which allows to de-couple the intended model from the sequential order of the source text (TEI-Consortium 2016, sec.4.3.2).

A different way is a declaration like "'`<figure rend="float chapter">`', which allows a figure to float in the given limits.

HTML has no support for floats. Complex positioning operations are possible by (a) manipulating a DOM with ECMA script, and/or (b) the box model and positioning options of CSS.

OD-T has a rudimentary support for floats, by element types `<draw:frame>` and `<draw:floating-frame>` (Durusau and Brauer 2011, sec.10.4). The concept of "Frames" is interesting and unique: different representations of *the same conceptual contents*, among which an application or a user may choose. But the realisation is again bad design, it hurts compositionality (only arbitrarily selected elements from the "drawing" realm can be contained) and its use for floating objects hurts SoC (floating has nothing to do with graphics, Section 10.4.1 of the spec says "Within text documents, frames are also used to position content outside the default text flow of a document." This smells like a hack !-)

d2d_gp: the standard text model supports arbitrary many series of floating objects, with user defined kinds, labels and captions. Its XHTML/CSS/ECMA-script back-end implements complex navigation and ToC generation with expand/collapse.

XML in general does know nothing about floats, because these live on the level of a concrete back-end of a concrete XML instance.

3.9 References Into the Text

Frequently, a text wants to refer to (a) a different text as a whole. Or to (b) to a segment of text, or to (c) to some single point in the text flow, i.e. a *text position*. Cases (b) and (c) can refer to (b0/c0) the referring text itself, or (b1/c1) to a different text, as in case (a).

Case (a) is easily done via citations, a kind of entity, see Section 3.5. In the other cases additionally the *human readable textual position description* must be supplied, plus a *computer readable technical position encoding*, which allows e.g. an interactive viewer to jump to that location.

When referring to segments of characters (cases b0/b1), the difference between tree-free and tree-bound becomes relevant again, see Sections 2.7 and 3.4 above.

The textual position description can be synthesised from the technical position encoding as long as the segment corresponds to a text component which has a numeric coordinate plus a kind name. This yields texts like "section 2.3.1.4" or "Figure 7.1". (For this synthesis, the *natural language* of the document must be known, which can be non-trivial in case of multi-lingual documents.)

Addressing a text component of some finer granularity, or addressing an exact position in the flow of text (cases c0/c1), can in most cases only be achieved by an additional "manual" counting of paragraphs and sentences, as in "last sentence of last paragraph in section 2.3.1.4". As soon as this happens, the exact borders of paragraphs and sentences, else an informal property, suddenly become relevant. This can be non-trivial, especially for tree-free segments.

Referring to *page numbers* and *page relative line numbers* should normally be avoided, because it is specific for one particular paper rendering, which is in most cases not considered substantial, i.e. not part of the text model. ("This practice is not generally recommended [..] since the pagination of a particular printed text is unlikely to be of structural significance." (TEI-Consortium 2016, sec.3.8.1))

A different case is when the page numbers of an important ancient printing T0 of the base text T1 are contained in the editors annotation layer of a modern edition T2, as the page numbers of the Bekker printing of Aristotle's "Categories" (Kalvesmaki 2014) or the page numbers of the Rosenkranz edition of the "Kritik der reinen Vernunft". In this case the historic page numbers of T0 are part of the model of T2 and rendered inline, as superscripts or sim.

Additionally, many historic text corpora are supplied with a *canonical reference system*, the human readable form of which has been standardised in the humanities: for the Bible, the Iliad, for the works of Aristotle and Shakespeare, etc. Kalvesmaki (2014) gives a survey how to integrate them into digital media.

Most TMFs allow to insert *anchors* at arbitrary positions in the text flow. While not always a textual position description can be synthesised, a digital rendering always allows to "jump" to the corresponding rendered position.

All *anchor-relative* numeric coordinates and thus all synthesised position description are automatically adjusted when a text evolves, e.g. when a section is inserted. Contrarily, all manually constructed location descriptions can become invalid, like "in the preceding section" or "see above[!] in section xxx".

Pointing from T1 into the foreign document T2 would become much easier as soon as the anchor definitions of T2 would be considered substantial part of the model and thus "exported" and "visible from outside". Currently no digital TMF supports this. Analysing the "`.aux`" file generated by **LaTeX** is technically possible, but must be considered "a hack": the stability of its contents is not guaranteed by the author of the model.

Manuscript and **Typescript**: references can be made to section titles and numbers, or to arbitrary text positions, in which case the reference text must be constructed manually. These links can be put in backward and forward direction. References to page numbers can (in most cases) only be put backwards; forward references must be *back-patched.*

LaTeX is not very consequent: The same anchor defining command

`\label{α}`

can be used in very different contexts, binding the user supplied identifier α to very different position informations: appearing in a caption of a floating object it binds to the numeric coordinate of the containing float; in main text to that of the nearest enclosing section; in a footnote it binds to its sequence number. The complementary `\ref{α}` command returns only that numeric coordinate, but all text (including the kind name) must be prepended manually by the user, which is error prone, e.g. when confusing "chapter" and "section". The `\pageref` command returns the page number of the same position. Most informations are correct not before the second or third run of the LaTeX processor.
Contrarily, the library package `varioref` tries to synthesise natural language page identifications, like "on the next page", taking into account the main human language of the text. There are few cases where this method does not find a fixpoint, the so-called "varioref jitter" (Goossens and Mittelbach 2004, pg.68).
Citing external texts is supported by "BibTex", which is a chapter of its own. Co-ordinates in the inner of these external texts could be generated and rendered by analysing their "`aux` files", but we do not know of any package supporting this kind of trans-document references.
An additional package (`hyperref`) allows to insert hyperlinks in the result, iff the PDF back-end is used (pg.78).

Lout is more systematic than **LaTeX** and defines `@PageMark/@PageOf` as label/ref pair for page numbers of flow text. The general `@Tag` can be inserted into structures, like sections, figures, tables, and `@PageOf/@NumberOf/@TitleOf` allow to retrieve these different values. Automated and explicit generation of hyperlinks is supported iff the PDF back-end is used (Kingston 2013, pg.53pp).

DocBook provides element tags `<anchor>, <xref>, <link>, <olink>`, which span a net of cross references. The rendering is implementation dependent. All higher-level elements of the text model can be used as targets for the references by giving them an `@xml:id` attribute. The element `<xref>` generates the reference text automatically (e.g. "figure number 1.2"), while `<link>` requires the visible text defined in the text model explicitly (Walsh 2010, sec.3.7.2).

TEI supports the full range of XPointer addressing with its `<ref>` element, see Section 3.4 above. This allows free navigation, but only along the DOM's technical tree structure. Contrarily, the `@cRef` attribute (allowed only for few selected element classes) realises a *canonical reference*, which is "any means of pointing into documents, specific to a community or corpus." (TEI-Consortium 2016, sec16.2.5). Kalvesmaki (2014) describes the accompanying CITE/CTS project, which targets at more precise syntax definitions and HTTP-based implementation of canonical references.

HTML only has anchors `<a id=...>..</a>` as its positioning device. ECMA script and CSS allow the evaluation of XPath expressions, which allow to identify any position in a document.

OD-T has an elaborated concept for defining anchors and referring to documents and anchors. The definition is even tree-free and has been discussed above in the context of segmentation, see Section 3.4.
There is no support for bibliographic data base, or for rendering fine granular co-ordinates of a citation.

d2d_gp has a complex architecture for synthesising absolute and relative reference texts, like "in this list, item number 7" or "in the preceding section, second list, item 2.1, second table, row 3, column 5".

XML in general does know nothing about user readable text coordinates, because these live on the level of a concrete XML instance.
The accompanying standard XPath is a language to address all components of an XML encoded text body, elements, attributes and character data, by an almost compositional expression language. This is a natural candidate for a back-end for text references, together with the `@xml:id` mechanism mentioned above.

3.10 Compositionality and Page Breaks

The mapping of the (mostly linear) character order to the pages of a printed copy can be considered part of the text model or not, as discussed in Section 1.3 above. Page breaks are contained e.g. in the models of programmed instruction books, see Section 4.2. If not, the TMF must insert page breaks automatically whenever rendering a longer text for printing onto a sequence of paper sheets. This is controlled by style specific rendering rules, like "Avoid page breaks in the midst of a poem" or "Clear page before a chapter".

Basically, the text model as such and the needs of the rendering process should be coupled as little as possible. But in practice this independence is not yet totally realised.

An important exception is e.g. the printing of **LaTeX**-tables which stretch over more than one page: Dedicated variants must be chosen by including packages like supertabular aut sim., which allow printing but have other restrictions (Goossens and Mittelbach 2004, pg.256).

There are other examples of these interdependencies, e.g. when printing graphics which are embedded in **HTML**. All these cases can be seen as a violation of compositionality: the choice of the tabular implementation is not longer independent from its application context, here: printing to a particular paper size.

Future evolution of software systems must aim at eliminating these dependencies, thus further clarifying the difference between model and rendering, between substance and accidentals, and augmenting the reign of compositionality.

4 Text and Time

The preceding sections described a statical model of text. The *temporal dimension* is a further orthogonal axis to that grid; its consideration raises philosophical questions about the essence and identity of text, as discussed by Huitfeldt, Vitali, and Peroni (2012). Most tasks in Digital Scholarship are temporal activities and related to temporal phenomena, see the standard textbook by Pierazzo (2015). The following section is again restricted to the technical questions and analyses few selected important use cases, with similar devices as in the preceding section.

4.1 Dynamic Physical Appearance (DPA) of a Text

One of the most important new features of digital text processing, beside automated comparing, indexing and search, is the *Dynamic Physical Appearance (DPA)* of a text rendering, on general purpose computer displays. (This section analyses the mere technical aspect; the very different applications are discussed in the following sections.) One and the same text model can be presented in very different views, controlled directly or indirectly by some user, e.g. the reader or the author or the editor or some software system.

There are many different applications of this feature. In principle, simple scrolling on a computer screen, and searching and highlighting of found occurrences by a simple PDF viewer is already a dynamic appearance. More elaborate is the *expand/collapse mechanism* found on HTML input forms, or with program source text editors from some Integrated Development Environments (IDEs).

The possibility for these "physically dynamic" ways of presentation may (a) live *totally outside* of the text model, e.g. when a dynamic way of rendering is applicable to *any* contents, as long as it is encoded physically in the corresponding format.

Dynamic appearance (b) may be *foreseen* in the text model, as far as tool tips, captions, paragraphs and links of selected kinds are meant to pop up, collapse, expand, serve as jump targets, etc.

It may (c) be *totally controlled* by the model, i.e. part of it in the narrow sense, if the model controls its outer appearance by a kind of *musical score*, which applies a series of transformations according to a fixed sequence of durations, or a by a *state machine* which reacts on user clicks, or by a combination of both. This is often called *animation*.

This distinction partly goes parallel to the distinction between two different sources of activity: the trigger for change can come from (I) an input activity from the user, or (T) the elapse of a certain time interval, after some preceding trigger.

Furthermore, changes can happen (α) gradually or (β) step-wise.

The following use cases employ different combinations of these alternatives.

4.1.1 Dedicated Hardware for Dynamic Text

A relatively new medium for presenting text are single-line LED displays which scroll a text from right to left, so that long sentences are readable in segments. Public transport companies use these in case of incidents to inform their passengers, (The type of DPA is aαT when scrolling or aβT when switching.) This reading situation clearly shows that this way of presentation induces a new kind of reception, which should be taken into account by the authors, but hardly is.

The principle of dynamic text rendering is much older, namely from the nineteen-twenties. One source is the art of composing *movie title sequences*; another very different origin are the dynamically switched neon bulbs in *illuminated advertising* (“Time Square illumination”). Of course the latter were “hard-wired” and could only change between few variants. The contemporary version of this are the typical LED displays showing the state of a Späti door in Kreuzberg of type cβT, see **Figure 4**.

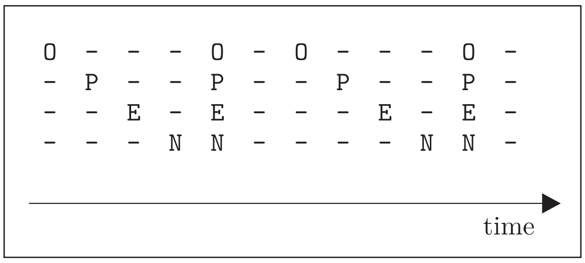

Figure 4: Score notation for an LED Display.

Manuscript and **Typescript** can appear in a dynamic way when physical layers are analysed by some *X-ray process*. This a common, but exceptional situation, since the dynamic appearance is not used to *present* different layers of the text model, but to *construct* (re-construct, explore, ...) them. The type of DPA is bβI.
Dynamic appearance may be reached by *scans/photographs* of the original text. This is the basis of all techniques for movie title trailers in pre-digital times. Nowadays scan data can be presented by digital processing in a dynamic way, as it is often done on websites which publish important historic documents.
But in visual arts much older methods to animate texts can be found:

Manuscript and **Typescript** are magnified and copied to transparent or semi-transparent material and then subject to changes of the illumination, which brings dynamic changes of visibility, thus contents, thus meaning. The type is bαI or bαT.
Similar are the simple on-off dynamics from programmed instruction books:
The answer to a question is printed in black but covered with a confusing maze of red letters, – under red light the answer becomes readable (Type bβI).

LaTeX maps its output indirectly to "`.dvi`", which is translated to **postscript**, or directly to PDF, in more recent implementations. **Lout** generates **postscript** or PDF directly.
Both these back-end formats, and thus the TMFs, are per se not capable of dynamic appearance. The more recent implementations which produce PDF directly allow to embed hyperlinks, which at least allows dynamic *navigation*.
Nevertheless, the famous `NextStep` operating system in the 1990s employed (a special dynamic extension of) **postscript** as the basis of its operating system GUI, which is doubtlessly a dynamic application (Adobe Systems 1993). The necessary animation could only be achieved by frequent (and efficient) re-generation of the displayed pages. In this sense, **LaTeX** and **Lout** could also be employed dynamically.

LaTeX s packages for slide shows (see Section 4.5 below) implement an expensive work-around and produce for each dynamic slide a whole *series of static pages*, each of duplicates the constant parts redundantly, and which are projected one after the other. (Type bβI).

TEI contains temporal information in transcriptions of speech (TEI-Consortium 2016, sec.8.3.6, 8.4.2, 16.5.2); with "performance text" it models simultaneous execution (sec.7.2, 16.5.1). So *temporal information can be substantial part of* the text model.

DocBook, TEI and **d2d_gp** do in no way specify or restrict the rendering back-end, so dynamic presentation is possible, but not supported as first-class resident.

HTML does support physically dynamic rendering by actively changing the underlying *Document Object Model (DOM)*, via (a) the use of ECMA script (Type bβI and cβT) or (b) the temporal control structures from SMIL (Dailey 2010; W3C 2008) (Type cβT only). In combination with embedded SVG for graphics, which is typical for the application contexts described in the following, also type α = gradual changes, are possible.

OD-T supports animation elements according to SMIL (Durusau and Brauer 2011, sec.15), (W3C 2008). Again, SoC is violated since animation means are duplicated in different contexts by `<presentation:animations>, <draw:anim>` `<anim:..`, etc. They support interactive animations (type bI, via event handlers) as well as automated animations, controlled by duration values (type cT).

XML in general: SMIL is a generic concept for temporal changes, either smooth or stepwise, either interactively/user controlled or automated, and has been adopted to many different XML encoded model languages, including SVG. (W3C 2008; Dailey 2010)

4.2 Interactive Reading (IR)/Computer-Aided Reading (CAR)

According to our initial definitions (see Section 1.3), a reading person cannot change the identity of the text, but its rendering. This we call *Interactive Reading (IR)*.

An interesting historic pre-digital variant were the printed but interactive books for *programmed instruction on paper* = *programmed instruction books*, developed in the Sixties of the last century. Here the next page to read is presented according to some input of the pupil in the preceding step. This can simply mean to select between several page numbers, but can even include the automated generation of a "new" text, in the sense of our definitions, reflecting the user's input. (DPA type is bβI) Due to these books, Computer-Aided Reading (CAR) (which is always interactive) is only a proper subset of IR.

Nowadays, these teaching systems are of course computer based, use all types of DPAs as described in the preceding section, and thus allow a much finer granularity, a much larger number of variants and the integration of other media like sounds and movies.

A most simple case of IR/CAR is a *tool tip text* which appears on a screen when the user's pointing device touches the area of the visual rendering of a particular text component, and vanishes after a time out or when leaving this area (type bβI or bβI+T). Normally the tool tip text gives further information about an entity (cf. Section 3.5) or about a clickable jump target.

The nowadays popular *e-book reader* could enhance the possibilities of reading: even in fine arts, an electronic index could be very helpful. E.g. for keeping track of the more than two-hundred characters in "War and Peace". One of the author prefers reading paper versions, but often misses an *accompanying* electronic index or search function, e.g. to retrieve a particular situation in the "Recherche" or an aphorism on a particular topic by Nietzsche.

CAR becomes esp. important to present multi-layer text, and to present time related structures contained in the text model as such, as discussed in the next sections.

In general, CAR can be applied to present to all components described in the preceding sections in a more or less sensible way: collapse paragraphs of a certain kind, highlight entities and references, highlight table cells, columns and rows, etc. An implementation allowing this for arbitrary text models (of a particular meta-model) is of type aβI.

The authors have programmed an SVG based *import graph*, which makes invisible all edges but those connected to the node currently pointed to (Lepper and Trancón y Widemann 2010, section "Containment Graph"). So very dense graphs can be made useful for the reader for information retrieval; this is a rather special variant of IR of type aβI.

André and Pierazzo (2013) describe the project for an interactive presentation of pages of a notebook by Proust, which visualises the historical temporal process of writing and editing by the temporal process of presentation. Similar ways of rendering will certainly be applied more and more in future.

When not only links are clicked but also text is entered (as in programmed instruction), then the border from IR to "*interactive writing*" may be crossed. When users fill out an empty form T1 given in PDF, then according to our definitions in Section 2.1 they create a new document (T2+T1).

For **Manuscript** and **Typescript**, dynamic appearance for a reader is only possible when scanned to some digital image format. In this context IR can be sensible to present the results of philologic research and the separation of layers by time and authors.

LaTeX and **Lout**: (n.a.)

DocBook does not specify a particular back-end. As candidate for dynamic presentation it provides several element types for cross references. A frequent processing strategy is the generation of **HTML**, which translates these references into links for navigation. Even more, descriptions of items can be realised as tool tip text, which pops up under the mouse pointer. Collapse/expand segments (by using additionally ECMA script/DOM transformations) are also rather frequent, see the **DocBook** documentation web site as its self-application (Walsh 2010).

Anyhow, in the genuine application area of **DocBook**, namely computer software documentation, CAR is highly sensible.

TEI is a very versatile and module based system, which allows to construct complex and stratified formats for text models. IR can be highly sensible when one model combines different angles and strata of information. Esp. the temporal informations, as described in sections 4.3 and 4.4, are candidates for interactive reading.

HTML: dynamic presentation controlled by some user input can be realised by dedicated ECMA script code, which manipulates the Document Object Model (DOM) according to user events. Expand/collapse is quite frequent, e.g. in the *wikipedia back-end*. Tool tips are first class residents and ubiquitous by the `@title` attribute. But they are *not compositional*: their possible contents are limited to a simple text string without any formatting.

OD-T supports conventional navigation as first-class resident. By user controlled animation (Durusau and Brauer 2011, sec.19.402) more IR could be implemented, but this looks tedious.

d2d_gp: IR is related to the back-end, thus not to **d2d_gp** as such. But it is extensively used by some elaborate instantiations of **d2d_gp**, e.g. in musicology (Lepper 2015).

XML in general does know nothing about interactive reading, because this lives on the level of a concrete back-end of a concrete XML instance. But the embedding of animation standards and ECMA script for explicit DOM manipulation is foreseen to define dynamic behaviour.

4.3 Historic Versions of the Same Text Document

Editors can model different temporal stages of a text T1 in course of their editing work, while creating the text T2. These temporal stages can be regarded as different versions of T1, or as substantially different and similar texts T1.1, T1.2, etc. Anyhow, the historic-temporal evolution of T1 is the real subject of the text T2, and T2 is the apparatus of a critical/genetic edition or even a larger linguistic study about T1.

This can be even more complicated, since also *different hypotheses on the temporal evolution of a text* can be the real subject of T2: we get a "tree with three-coloured vertices", representing evolution, inclusion and alternative theories.

Only **TEI** supports this explicitly, since it has been designed esp. for this use case, the encoding of historic text bodies. (As mentioned above, in all programmable systems like **Lout** and **LaTeX**, the necessary means can be realised by additionally plugged-in code.)

IR has already been employed successfully to present the different versions and layers of a historic text object. (André and Pierazzo 2013)

With **Manuscript** and **Typescript** the question for historic versions coincides with that of "work-in-progress versions", as discussed in the next section. Indeed, the *temporal order of text layers* can be a question of central interest: Esp. in musicology ("Has this correction been added by the composer or by the premiere conductor?"), or when trying to edit the unfinished multi-stage pages written by Hölderlin and Proust.
Here a temporal sequence of stages may even be insufficient, but a *multitude of hypotheses about different sequential orders* may be the adequate model. In this case, not "one(1) text" does exist "realiter", but a multitude of possible texts, each of which is a sequence of different stages. Then this multitude is the subject of scientific discourse, and thus the "reality" which must be modelled.

LaTeX and **Lout** do not have dedicated support, but of course can present different historic version with different renderings, controlled by macro programming. (The same holds of course for all other TMFs which support control of physical rendering and macro programming.)

DocBook does not support historic text versioning as first class resident, but the `<info>` meta information element, which can be attached to all higher level structural elements, allows to encode `<author>`, `<authorgroup>`, `<date>`, `<edition>`, `<releaseinfo>`, `<revhistoriy>`, etc., and could be used for this purpose (Walsh 2010, sec.3.5).

TEI has been designed focused on historic text bodies. Therefore there is extensive support for modelling historic aspects, e.g. the physical matter of manuscripts (TEI-Consortium 2016, sec.10.7-10.9) and all kinds of corrections, deletions, changes, insertions, etc. (sec.11.3), including the colour of the ink used and kind of damage suffered.
It is also the only TMF which allows to model the authorship of an arbitrary small sub-segment of the text model by the `@hand` attribute.

HTML has dedicated support of text modification by the tags `<del>` and `<ins>`.

OD-T has elaborate elements for *change tracking*, which are intended for volatile versions, but can perhaps be abused to model historic layers (Durusau and Brauer 2011, sec.5.5).

d2d_gp: the basic model does not support historic layers, but user instances easily can define the required element structures.

XML in general does know nothing about historic versions, because these live on the level of a concrete XML instance.

4.4 Volatile Versions When Creating a Text Document

A somehow dual situation is given when the versions of creating a fresh document must be handled explicitly. The history of versions and the process of evolution is not *part of* the final model, as in the preceding section, but required for *constructing* it. But it may be considered part of a *temporary text model*. The related components will be removed from the text as soon as the final model is completed, but are needed during its construction phase. All related issues become esp. important in the frequent case of a *multi-authors text*.

Typical components are *change bars*, and the mark-up of text fragment elimination and replacement, as long as a technical manual or a legal text is on its way through committees.

In this context the Source Text Strategy (SrcTS) is very useful, see Section 2.2 above, esp. in combinations with automated processing: All TMFs which follow

SrcTS allow (a) to insert comments into the source, which are not part of the model (i.e. of the final, intended model, the model under construction), (b) to apply text processing standard tools like `diff, grep, sed` to the source text, and (c) to use version control systems like `RCS` or its younger competitors for assigning and controlling responsibilities among the authors involved. For this, the support of the TMFs is rather limited, see details below.

The possibility to assign *individual modification rights* to different authors is implemented in PDF, but in none of the discussed TMFs. But **XML** in general does allow to implement it rather easily, when the act of "storing the model" would be defined on the level of DOM nodes, instead on the textual representation. To our knowledge, this mechanism has not yet been implemented in a generic way. It could be used immediately by all XML based TMFs (**DocBook, TEI, HTML, OD-T, d2d_gp**).

A slightly different case, but important in practice, is the addition of annotations on a separate, dedicated data layer, as it is implemented e.g. in the PDF framework. In this case a *second text* T2 is created, discussed above in Section 2.1 als multi-layered text. The author of T2 is the reader of the first text T1, and the structure of T2 is defined upon the structure of T1.

E.g., one of the authors defined a very efficient communication protocol between himself as composer and a contracted sheet music engraver: each issue was modelled by one PDF "annotation balloon" in the music score under creation, T1. The colours were assigned: one for each side for their initially entered comments, followed by a precisely defined state machine for the issues' resolution process, reflected in colour changes.

With **Manuscript** and **Typescript**, the historic layers and the volatile layers mostly coincide, see Section 4.3. An exceptional case is reported by Goethe, who inserted white paper sheets for the missing scenes into the growing manuscript of "Faust II", for to get a haptical feeling for the work still to do, and thus additional motivation. These empty pages have been replaced by written ones, so they are part of the volatile, but not of the historic layers. (Talk to Eckermann, 17.Feb.1831) (Goethe 1986, pg.463).

LaTeX does support dynamic creation aspects by the additional package `version`, which allows text fragment combination on the source text level under program control.
The normal glyph renderings supports things like `strikethrough`, which can be used for version indication by explicit macro programming. A dedicated package adds `changebars`.

Lout's macro programming could realise something like **LaTeX**s `version` package. Stroke out characters and margin paragraphs can be employed to represent an editing history.

DocBook provides `sidebar`, which can be used as change marker (Walsh 2010, sec.3.6.12). As described in Section 4.3, the meta info element `info` can be used for temporal information.

TEI uses `@resp` (TEI-Consortium 2016, sec.11.3.2.2) and `<respons>` (sec.21) to indicate the person responsible for particular decisions in the mark-up process, and `@cert` (sec.11.3.2.2) and `<certainty>` (sec.21) to indicate the certainty of a particular decisions. By these means, meta information becomes part of the document model.
One may use `@note` (sec.3.8) also for meta-information, describing the current state of a transcription process.

HTML has dedicated support for the rendering of text modification by the tags `<del>` and `<ins>`.

OD-T has elaborate elements for "change tracking" (Durusau and Brauer 2011, sec.5.5).

d2d_gp supports volatile versions by the source text strategy (SrcTS).

XML follows SrcTS, which allows the application of *version control systems* with automated change reporting. It supports comments which can be used in a text construction process. Individual modification rights for collaborating authors could be defined in a generic way.

4.5 Animated Text in Slides and Presentations

Animated text for slides is not supported directly by most TMFs, but by some sibling software. So microsoft offers the (in-)famous "power point" as part of their "office" package, a sibling to "ms word", and **OD-T** has a zoo of `<presentation:..>` elements in parallel to its text model.

The `beamer` package (Tantau, Wright, and Miletić 2011) and some older predecessors (`seminar, prosper, FoilTex,` etc.) are plug-ins for **LaTeX**.

Slides are "dynamic" by definition and in their ancient realisation, because they are a sequence of images presented to the public by projection, one after the other, ordered in time, following the tempo of a life speaker (DPA type bβI) or a prefabricated tape (bβT).

This can be (and historically has been) limited to a mere sequence of static picture contents. For our subject we have to ask for *dynamic behaviour of the text contents* of one single such slide.

Here we have different features:

In "Power Point", everything can "fly in", "fade out", "crumble" or "explode" (DPA type cαT). But only by selecting from a fixed set of "effect generators", which offer a fixed sets of parameters each, and basically operate on pixel graphics. There is no principle limit, but a practical, since the particular effect generators must be available.

The `beamer` **LaTeX** package allows arbitrary change of text content, controlled by a concept of "phases", e.g. replacing one text fragment by another. An instance of this is the semi-automated change of visual aspects, for fading-in of the items of a list one by one, etc. The temporal definitions have the granularity of stages (DPA type bβI); *smooth* moving, dynamic fading and exploding are not supported.

With all **XML** based formats, the combination of the extensions *Scalable Vector Graphics (SVG)* (W3C 2011) and the animation features of *Synchronized Multimedia Integration Language (SMIL)* (Dailey 2010; W3C 2008) can be used. These allow in principle all kinds of temporal behaviour: position, rotation, colour, transparency, font size, etc. of each text fragment can be changed gradually over some time interval, in arbitrary parameter value curves. (Generally DPA type bαT), but exchange of fonts can only happen stepwise, and exchange of text contents only by switching on and off the visibility of different text objects (both of type β). This approach follows Compound Source Strategy (CmpSS): Text stays text, even under graphic transformations, and thus can be subject to search and replacement.

The construction of these temporal structures is awesome, and misses "text quality" (in the sense of a "direct reification" of a mental model), because there is no simple direct relation between the intended time functions and the demanded expression syntax.

So we have three fundamental different strategies, with different pros and cons each:

1. Arbitrary and smooth image transformations, on all parameters, and even on pixel level, but only by predefined processors ("powerpoint").
2. Non-smooth exchange of text components and of all text rendering parameters, without moves (**LaTeX**).
3. Smooth image transformations, not arbitrarily on pixel level (no "blur" nor "crumble"), but on most text rendering parameters (colour, size, transparency, position and rotation) by SVG+SMIL.

The constructions of these transformations, in the current implementations, have again very different grades of flexibility:

1. No means for abstraction and automation, only interactively definable and stored in an opaque binary format ("powerpoint").
2. Fully under program control, easily abstractable and convenient usage (**LaTeX**).
3. Fully under program control and abstractable (ECMA script, DOM model, etc.), but counter-intuitive expression language, very hard to write down manually (SVG+SMIL).

Interestingly, the types of supported triggers are complementary:

1.+2. The frameworks intended for slide shows ("powerpoint" and **LaTeX** packages) basically react only on user input.

3. The (SVG-SMIL) animations are executed according to fixed duration values, defined in the source text, i.e. part of the text model. User input triggering requires explicit additional coding (manipulating the DOM with ECMA script, etc.) But coding allows also the combination of both trigger sources.

Manuscript and **Typescript**: (n.a.)

LaTeX supports slides generation by dedicated packages, see discussion above.

Lout (We do not know about an additional package for slide generation, but of course this would be feasible.)

DocBook has no genuine relation to slide generation, but of course processors (which are not defined in the standard) could extract e.g. the elements dedicated to program documentation (like `<programlisting>, <computeroutput>, <cmdsynopsis>`) for integrating them into a slide presentation.

TEI models can contain explicit temporal information, see Section 4.1 above. Translations to SVG/SMIL etc. could present the text accordingly.

HTML: as with interactive reading (see above), dynamic presentation controlled by some user input can be realised by dedicated ECMA script code, which manipulates the DOM according to user events.

OD-T does not support slides generation, but a sibling application does, see general remarks above. The corresponding elements are in the namespace "`{urn:oasis:names:tc:opendocument:xmlns:presentation:1.0}presentation`", (Durusau and Brauer 2011, sec.9) A possible toplevel element chain is

```
<office:presentation><draw:page><draw:rect><text:p> …
```

which allows a very restricted kind of compositionality. (The other way round every text may contain arbitrary graphic elements, see Section 3.3.3.)

d2d_gp does not support slide generation. (A user defined extension is of course possible, but not intended yet.)

XML in general does know nothing about slide show production, because this lives on the level of a concrete back-end of a concrete XML instance. (We do not know of any attempt to model slides, but of course this is possible in principle.)

4.6 Animated Text in Visual Arts

In *visual arts* the animated presentation of text has a history of nearly hundred years, starting with the graphics of movie title sequences, as mentioned above. A more abstract, more recent and explicitly artificial example has been the video "Sign o' the Times" by Prince in 1987. (Nilsen 2004, pg.623, acc.to wikipedia)

As mentioned in the previous section, the combination of XML plus SVG plus SMIL has a very satisfying range of effects as its outcome, but cannot be denotated in an intuitive way. Therefore the authors wrote translation software based on their **tscore** time representing formalism, for easy denotation of temporal artistic text processes. The result has been published as a combination of source text, binary application, example score data, generated output, i.e. animated typographic art, and documentation (Lepper and Trancón y Widemann 2013a, 2013b).

Of course, the translation code is *dedicated* to one particular setting of input score format and intended output animation style. The programming language (here: Java) is the means for defining and realising the relation between these two. But this specific text needs only two to three pages, it is easily adopted to other intentions.

Manuscript and **Typescript** do appear in animated form in visual arts quite frequently, nowadays mostly by scanning and projection. See Section 4.1 for pre-digital methods for animating text objects.

$\LaTeX$ and **Lout** are suited for this purpose only in the limits discussed for slides, see preceding section.

DocBook (n.a.)

TEI (n.a.)

HTML: The physically dynamic rendering is realised by SMIL expressions, which directly contain HTML or SVG expressions, for fixed scheduled execution. (In contrast to the control by user events and ECMA script in the preceding use cases.)

OD-T supports dynamic rendering by SMIL animation, as described above. So in principle it can be used for dynamic texts in arts. We assume that programming will be tedious, and that not many client programs will render correctly.

d2d_gp: (not supported in basic model; model definition possible.)

XML in general does know nothing about animated text in visual arts, because this lives on the level of a concrete back-end of a concrete XML instance.

4.7 Online Information Systems

Even the "identity of a text itself" (as defined in the fundamental discussion above, see Section 1.3) can be variable. This is the case with *online information systems*: e.g. the current prices on a stock exchange are often presented on screen in tables, in one column the company's name, in the next the figures changing in real time. But of course these figures could also appear in a flow text, changing also the wording of the sentences and the colour of the mark-up dynamically, replacing a bull icon by a bear.

In any case, these online information systems require to support dynamic appearance, as defined in Section 4.1.

Manuscript and **Typescript** (n.a.)

LaTeX and **Lout** are suited for this purpose only in the limits discussed for slides, see Section 4.5.

DocBook does not define the processing of the text model. Therefore dynamic elements can be integrated without fundamental problems: e.g. a sequence made of `<userinput>, <replaceable>` and `<computeroutput>` can be rendered as an **HTML** input form, the activation of which leads to dynamic evaluation of the user input and a new visual output.

TEI mainly defines static text models, but few attributes with an "URL-like" data type indeed refer directly to an online resource, as in `<keywords scheme="http://id.loc.gov/authorities/about.html#lcsh">`... `(TEI-Consortium 2016`, 2.4.3), or in `<w lemmaRef="http://lexicon.org/latin.xml#danaii">Danaos</w>`
(TEI-Consortium 2016, 17.1.1).

HTML can be used for online information systems, either by its own dynamic features (ECMA script operating on DOM, as described above), or by embedding "objects" or "applets" which execute dynamic display of data, related to the surrounding **HTML** text only for positioning/scrolling/etc.

OD-T: while animating arbitrary text looks tedious and thus deprecated (see Section 4.1 above), the presentation of dynamic data input is first class resident. For this "dynamic data exchange" dedicated elements are foreseen, namely `<office:dde-source>, <office:dde-application>, <table:dde-link>, <text:dde-connection-decl>`, etc. But the rendering of the dynamically received data information is intended to be done by the spreadsheet program, not as part of dynamically rendered text.

d2d_gp: (not supported in basic model; model definition possible.)

XML in general plays a fundamental role in online information systems, because nowadays the format of many data sources is XML encoded. So this data is "text" in a technical sense, but in almost all cases treated like "binaries" and translated into a very different format for presenting it to humans.

5 Comprehensive Table of the Characteristics of the Different Text Modelling Frameworks

The following **Tables 1** and **2** shows in short notation the most significant results from all preceding sections. The meanings of the entries are ...

YES	=	unconditionally supported
(y)	=	can be implemented using other features
NO	=	not supported in any case
(n)	=	normally not supported
–	=	not applicable
(src)	=	can be realised on source text level.
Q0	=	requirement is fulfilled badly.
upto		
Q5	=	requirement is fulfilled optimally.

Remarks in the table:

×1	=	SVG elements supported
×2	=	some proprietary graphic languages
×3	=	induced by the MSSPP
×4	=	when using the "CALS table" variant
×5	=	requires CSS with version ≥ 3
×6	=	by use of CSS classes
×7	=	by use of style definitions
×8	=	by use of SVG and SMIL
×9	=	in the very special case of "programmed instruction"
×10	=	list numbers for all items may be given explicitly

Table 1: Survey of Characteristics I.

	section	Ms.	Type.	LaTeX	Lout	DocBook	TEI	HTML	OD-T	d2d_gp	XML
Source Text Strat. (SrcTS)	2.2	–	–	YES	YES	YES	YES	YES	no	YES	YES
XML based	2.2	no	no	no	no	YES	YES	YES	YES	YES	–
Turing complete	2.2	no	no	YES	YES	no	no	no	YES	no	–
Multi-Layer/Author	2.1	YES	YES	(y)	(src)	(src)	YES	(src)	–	(src)	(src)
Text Graphic Art	1.7	YES	YES	YES	YES	no	no	YES (×6)	no	no	–
Compound Source Strategy (CmpSS)	2.4	–	–	Q5	Q1	Q1 (×1)	Q3 (×2)	Q1	–	Q4	Q5
Separation of Concerns (SoC), resuability	2.5	–	–	Q5	Q1	Q1	Q3	Q1	Q2	Q4	Q5
Compositionality	2.6	–	–	Q4	Q4	Q1	Q2	Q2	Q2	Q4	Q4
Tree free segments and highlighting	2.7, 3.4	YES	YES	(n)	(n)	YES	YES	NO	NO	(n)	(n)
Numb. of titles per section	3.1	n	n	2	1	1	n	n	n	2	–
Properties of the explicit hierarchy	3.1	1,3(×3)	1,3(×3)	1–4,7	1–4,7	1–4,7	none	1	none	1–4,7	–
Automated ToC generation	3.1	NO	NO	YES	YES	YES	YES	NO	YES	YES	–
Numbering scheme variable	3.1	YES	YES	YES	YES	NO	YES	–	YES	(n)	–

(Contd.)

	section	Ms.	Type.	LaTeX	Lout	DocBook	TEI	HTML	OD-T	d2d_gp	XML
Paragraph kinds definable?	3.2	YES	YES	YES	(n)	NO	YES	(y) (×6)	(y) (×7)	YES	–
In-paragraph displays?	3.2	YES	YES	YES	(y)	YES	YES	(y) (×6)	YES	YES	–
Margin paragraphs?	3.2.1	YES	YES	YES	YES	YES	(y)	(y) (×6)	no	(no)	–
Inline lists	3.3.1	YES	YES	(n)	(n)	(n)	YES	(n)	NO	(n)	–
List start no. adjustable	3.3.1	–	–	no	YES	YES	(y) (×5)	y (×5)	(y) (×10)	YES	–
Free bullet symbol	3.3.1	–	–	no	YES	(y)	YES	y (×5)	YES	YES	–
ListSubP	3.3.1	–	–	no	(n)	NO	NO	NO	NO	YES	–
Tables follow SrcMC	3.3.2	–	–	YES	YES	NO (×4)	NO (×4)	YES	YES	YES	–
Table-L and table-I cells	3.3.2	YES	YES	(n)	(n)	YES (×4)	YES (×4)	NO	NO	(n)	–
Alignment across rows	3.3.2	YES	YES	YES	(n)	YES (×4)	YES (×4)	(y) (×5)	NO	NO	–
Diagrams in same source (CmpSS)	2.4, 3.3.3	–	–	(y)	(y)	NO	(y) (×2)	(y)	(y)	(y)	–

Table 2: Survey of Characteristics II.

	section	Ms.	Type.	LaTeX	Lout	DocBk	TEI	HTML	OD-T	d2d_gp	XML
Multiple main languages	3.4.1	(y)	(y)	(y)	–	NO	YES	NO	NO	YES	NO
Lang. per paragraph	3.4.1	(y)	(y)	(y)	XES	YES	YES	YES	NO	YES	YES
Lang. per region/span	3.4.1	(y)	(y)	(y)	YES	YES	YES	YES	NO	YES	YES
Semantic categ. for entities	3.5	(y)	(y)	(y)	(y)	YES	YES	NO	(n)	YES	–
Automated index gen.	3.5	NO	NO	YES	YES	YES	YES	NO	YES	YES	–
Multiple indices	3.5	(y)	(y)	YES	YES	YES	YES	NO	YES	YES	–
Different index keyword sorting vs. appearance	3.5	–	–	YES	NO	(y)	YES	–	NO	YES	–
“Fuzzy” data; ranges of val.	3.5	YES	YES	NO	NO	NO	YES	–	NO	(y)	–
Hierarchical identifiers	3.6	(n)	(n)	(n)	(n)	(n)	(n)	(n)	(n)	(n)	–
Icons and formulas	3.6	YES	YES	YES	YES	(y)	YES	(n)	(y)	YES	–
Footnotes, Apparatus	3.7	YES	YES	YES	YES	(y)	YES	(n)	(y)	YES	–
Floating Objects	3.8	(y)	(y)	YES	YES	YES	YES	(y) (×6)	(y)	YES	–
Ref by full XPointer	3.9	–	–	NO	NO	NO	YES	YES	NO	NO	YES
Dynamic optic. appearance	4.1	(n)	(n)	no	no	–	–	YES	YES	(y)	–
Interactive Reading (IR)	4.2	YES (×9)	YES (×9)	(n)	(n)	(y)	–	YES	(n)	(n)	–

(Contd.)

	section	Ms.	Type.	$\LaTeX$	Lout	DocBk	TEI	HTML	OD-T	d2d_gp	XML
Tool tips	4.2	NO	NO	–	–	–	–	YES	–	–	–
Historic Versions	4.3	YES	YES	(n)	(n)	(y)	YES	(y)	(y)	(n)	–
Volatile Versions/Authoring Protocol	4.4	YES	YES	(y)	(y)	(y)	YES	(y)	YES	(src)	(src)
Indiv.Modifiation Rights	4.4	–	–	NO	NO	(n)	(n)	(n)	NO	(n)	(y)
Slide Show Support	4.5	(n)	(n)	YES	(y)	(n)	(n)	(y) (×8)	(n)	(n)	–
Suited for Dynamic Visual Arts	4.6	(y)	(y)	YES	(y)	NO	(n)	(y) (×8)	(n)	(n)	YES (×8)
Suited for Online Information Systems	4.7	NO	NO	NO	NO	NO	NO	(y)	YES	(n)	YES

Acknowledgements

Thanks to Peter Pepper, Berlin, for drawing our attention to the dynamic aspects.

Thanks to Kai Stalmann, Berlin, for valuable feedback.

Thanks to the anonymous reviewers who helped to improve the text substantially.

Competing Interests

The authors have no competing interests to declare.

References

Adobe Systems. 1993. *Display postscript system — introduction: Perspective for software developers.* Adobe Developer Technologies.

———. 1999. *Postscript language reference.* Addison-Wesley. ISBN: 0-201-37922-8.

André, Julie, and **Elena Pierazzo.** 2013. Le codage en TEI des brouillons de Proust: vers l'édition numérique. *Genesis (Manuscrits - Recherche - Invention)* 36 (Apr.): 155–161. DOI: https://doi.org/10.4000/genesis.1159

Bennett, Rick, Christina Hengel-Dittrich, et al. 2007. Linking die Deutsche Bibliothek and Library of Congress name authority files. *International cataloguing and bibliographic control* 36 (1): 12–19.

Bingham, Harvey. 2000. *Cals table model history.* https://web.archive.org/web/20110402011307/http://users.rcn.com/hwbingham/tables/calstbhs.htm (accessed Oct. 11, 2019).

Boyer, John. 2001. *Canonical XML 1.0.* Recommendation. W3C. https://www.w3.org/TR/xml-c14n (accessed Oct. 11, 2019).

Branden, Ron Van den, Melissa Terras, and **Edward Vanhoutte.** 2008. *TEI by Example.* Royal Academy of Dutch Language/Literature, etc. http://teibyexample.org (accessed Jan. 11, 2016).

Bray, Tim, Jean Paoli, et al. 2006. *Extensible Markup Language (XML) 1.1 (Second Edition).* Recommendation. W3C. http://www.w3.org/TR/2006/REC-xml11-20060816 (accessed Oct. 11, 2019).

Burnard, Lou, and **C. M. Sperberg-McQueen.** 2012. *Tei lite: encoding for interchange: an introduction to the tei: Final revised ediition for the tei p5.* Text Encoding

Initiative Consortium. http://www.tei-c.org/Vault/P4/Lite/teiu5_en.html (accessed Oct. 11, 1019).

Buzzetti, Dino. 2009. Digital editions and text processing. In *Text editing, print, and the digital world.* Ashgate, UK. ISBN: 978-0-7546-7307-1, https://www.researchgate.net/publication/290227303_Digital_editions_and_text_processing (accessed Oct. 11, 2019).

Dailey, David. 2010. *An svg primer for today's browsers: Chapter iv - smil animations embedded in svg.* W3C. http://www.w3.org/Graphics/SVG/IG/resources/svgprimer.html#SMIL_animations (accessed Oct. 11, 2019).

DocBook-Team. 2014. *Docbook 5.1 specification, rnc format.* DocBook.org.

Durusau, Patrick, and **Michael Brauer,** eds. 2011. *Open document format for office applications (opendocument) version 1.2.* OASIS. http://docs.oasis-open.org/office/v1.2/os/OpenDocument-v1.2-os.html (accessed Nov. 20, 2019).

Geschke, Chuck, and **John Warnock,** eds. 2006. *Pdf reference.* Six edition. Adobe Systems Inc.

Goethe, J. W. v. 1986. *Gesammelte Werke, Hamburger Ausgabe.* C.H.Beck. ISBN: 3-423-19038-6.

Goossens, Michel, and **Frank Mittelbach.** 2004. *The LATEX companion.* Second. Addison-Wesley. ISBN: 0-201-36299-6.

Huitfeldt, Claus. 1994. Multi-dimensional texts in a one-dimensional medium. *Computers and the Humanities* 28: 235–241. DOI: https://doi.org/10.1007/BF01830270

Huitfeldt, Claus, Fabio Vitali, and **Silvio Peroni.** 2012. Documents as timed abstract objects. In *Proceedings of Balisage: The markup conference 2012.* DOI: https://doi.org/10.4242/BalisageVol8.Huitfeldt01

IFLA Study Group on the Functional Requirements for Bibliographic Records. 1998. *Functional requirements for bibliographic records.* FLA Series on Bibliographic Control 19. München: K.G. Saur Verlag.

Ingarden, Roman. 1960. *Das literarische Kunstwerk.* Max Niemeyer.

———. 1962. *Untersuchungen zur Ontologie der Künste.* Max Niemeyer.

Jacob, Pierre. 2019. Intentionality. In *The Stanford encyclopedia of philosophy,* Spring 2019, ed. Edward N. Zalta. Metaphysics Research Lab, Stanford University. https://plato.stanford.edu/archives/spr2019/entries/intentionality/ (accessed Oct. 11, 2019).

Jannidis, Fotis. 1997. TEI in der Praxis. In *Jahrbuch für computerphilologie.* München: Computerphilologie Uni München. http://computerphilologie.uni-muenchen.de/praxis/teiprax.html (accessed Oct. 11, 2019).

Kalvesmaki, Joel. 2014. Canonical references in electronic texts: Rationale and best practices. *Digital Humanities Quarterly* (Boston VA) 8 (2).

Kingston, Jeffrey H. 1992. The design and implementation of the lout document formatting language. *Software—Practice & Experience* 23 (9). DOI: https://doi.org/10.1002/spe.4380230906

———. 2000a. *An expert's guide to the lout document formatting system.*

———. 2000b. *The Lout homepage.* http://savannah.nongnu.org/projects/lout (accessed Oct. 11, 2019).

———. 2013. *A user's guide to the lout document formatting system (3.40):Version 3.40.* http://download.savannah.gnu.org/releases/lout/lout-3.40-user.ps.gz (accessed Oct. 11, 2019).

Kittelmann, Jana, and **Christoph Wernhard.** 2016. Knowledge-based support for scholarly editing and text processing. In *Dhd 2016 – digital humanities im deutschsprachigen raum: modellierung – vernetzung – visualisierung. die digital humanities als fächerübergreifendes forschungsparadigma. konferenzabstracts,* 178–181. Duisburg: nisaba verlag. http://arxiv.org/pdf/1908.11135 (accessed Oct. 13, 2019).

Lamport, Leslie. 1986. *Latex user's guide and document reference manual.* Reading, Massachusetts: Addison-Wesley Publishing Company.

Lepper, Markus. 2015. Gustav Mahler, Dritte Sinfonie, Erste Abtheilung: Eine Annäherung. *senzatempo.* http://senzatempo.de/mahler/gmahler_sinf3_satz1.html (accessed Oct. 11, 2019).

Lepper, Markus, and **Baltasar Trancón y Widemann.** 2010. *D2d gp generated user doc.* http://www.bandm.eu/metatools/docs/usage/d2d_documentation_basic.deliverables_user_en/index.html (accessed Oct. 11, 2019).

———. 2013a. *Example instances of the TScore projekt infrastructure.* http://markuslepper.eu/sempart/tscoreInstances.html (accessed Oct. 11, 2019).

———. 2013b. Tscore: Making computers and humans talk about time. In *Proc. keod 2013, 5th intl. conf. on knowledge engineering and ontology development,* 176–183. Portugal: instincc, scitePress. ISBN: 978-989-8565-81-5, https://www.researchgate.net/publication/262057663_tScore_Making_Computers_and_Humans_Talk_About_Time (accessed Oct. 11, 2019).

———. 2018. Rewriting for parametrization. In *Tagungsband des 35ten jahrestreffens der gi-fachgruppe programmiersprachen und rechenkonzepte,* 51–67. Vol. 482. IFI Reports. University of Oslo. ISBN: 978-82-7368-447-9, http://urn.nb.no/URN:NBN:no-65294.

Lepper, Markus, Baltasar Trancón y Widemann, and **Jacob Wieland.** 2001. Minimze mark-up ! – Natural writing should guide the design of textual modeling frontends. In *Conceptual modeling — er2001.* Vol. 2224. LNCS. Springer. http://markuslepper.eu/papers/er2001.pdf (accessed Oct. 11, 2019). DOI: https://doi.org/10.1007/3-540-45581-7_34

Leunen, Mary-Claire van. 1992. *A handook for scholars.* Oxford.

Nilsen, Per. 2004. *The vault — The definitive guide to the musical world of Prince.* Nilsen Publishing. ISBN: 91-631-5482-X.

Pierazzo, Elena. 2015. *Digital scholarly editing : Theories, models and methods.* Routledge. ISBN: 1-472-41211-7. DOI: https://doi.org/10.4324/9781315577227

Renear, Allen H., and **Karen M. Wickett.** 2009. Documents cannot be edited. In *Proceedings of balisage: the markup conference 2009.* Vol. 3. Balisage Series on Markup Technologies. DOI: https://doi.org/10.4242/BalisageVol3.Renear01

———. 2010. There are no documents. In *Proceedings of balisage: The markup conference 2010.* Vol. 5. Balisage Series on Markup Technologies. DOI: https://doi.org/10.4242/BalisageVol5.Renear01

Rosenmann, Mauricio. 1995. *Chile o el p/fisco sauer (aus der sinfonía para nombres solos).* Pfau Verlag.

———. 1996. *Formicación.* Pfau Verlag.

Siemens, Ray, Teresa Dobson, et al. 2011. Hci-book? perspectives on e-book research, 2006–2008 (foundational to implementing new knowledge environments). *Papers of the Bibliographical Society of Canada/Cahiers de la Société bibliographique du Canada* 49 (1).

Tantau, Till. 2015. *TikZ and PGF.* http://mirrors.ctan.org/graphics/pgf/base/doc/pgfmanual.pdf (accessed Oct. 11, 2019).

Tantau, Till, Joseph Wright, and **Vedran Miletić**. 2011. *The beamer class – User guide for version 3.36.* ctan.

Taupin, Daniel, Ross Mitchell, and **Andreas Egler.** 2002. *MusixTEX — Using TEX to write polyphonic or instrumental music.* http://mirrors.ctan.org/macros/musixtex/doc/musixdoc.pdf (accessed Oct. 11, 2019).

TEI-Consortium. 2013. *Getting started using TEI.* http://tei.oucs.ox.ac.uk/GettingStarted/html/ (accessed Oct. 11, 2019).

———. 2016. *TEI P5: Guidelines for Electronic Text Encoding and Interchange.* http://www.tei-c.org/release/doc/tei-p5-doc/en/Guidelines.pdf (accessed Oct. 11, 2019).

Trancón y Widemann, Baltasar, and **Markus Lepper.** 2010. *The bandm meta-tools user documentation.* http://bandm.eu/metatools/docs/usage/introduction.html (accessed Oct. 11, 2019).

———. 2019. Simple and effective relation-based approaches to xpath and xslt type checking. In *Tagungsband des 36ten jahrestreffens der gi-fachgruppe programmiersprachen und rechenkonzepte,* 36–46. Vol. 488. IFI Reports. University of Oslo. ISBN: 978-82-7368-453-0, http://urn.nb.no/URN:NBN:no-75603 (accessed Dec. 10, 2019).

Walsh, Norman. 1999. *Xml exchange table model document type definition.* OASIS. https://www.oasis-open.org/specs/tm9901.html (accessed Oct. 11, 2019).

———. 2010. *Docbook 5: The definitive guide.* O'Reilly Associates. ISBN: 0-596-80502-0.

W3C. 2002. *An xhtml + mathml + svg profile: W3C working draft.* W3C. http://www.w3.org/TR/XHTMLplusMathMLplusSVG/ (accessed Nov. 20, 2019).

———. 2008. *Synchronized multimedia integration language (smil 3.0).* W3C. http://www.w3.org/TR/2008/REC-SMIL3-20081201/ (accessed Oct. 11, 2019).

———. 2011. *Scalable vector graphics (svg) 1.1.* 2nd. W3C, W3C. http://www.w3.org/TR/SVG11/ (accessed Oct. 11, 2019).

———. 2014. *Mathematical markup language (mathml) version 3.0.* 2nd. W3C, W3C. http://www.w3.org/TR/MathML3/ (accessed Oct. 11, 2019).

W3C HTML Working Group. 2002. *Xhtml 1.0 The extensible hypertext markup language: A reformulation of html 4 in xml 1.0.* 2nd ed. W3C Recommendation. W3C HTML Working Group. http://www.w3.org/TR/2002/REC-xhtml1-20020801 (accessed Oct. 11, 2019).

———. 2011. *Cascading style sheets level 2 revision 1 (css 2.1) specification: W3C recommendation.* W3C HTML Working Group. http://www.w3.org/TR/CSS2 (accessed Oct. 11, 2019).

———. 2011. *Css text level 3.* W3C HTML Working Group. http://www.w3.org/TR/2011/WD-css3-text-20110901 (accessed Nov. 20, 2019).

How to cite this article: Lepper, Markus, and Baltasar Trancón y Widemann. 2021. "Technical Topologies of Texts." *Digital Studies/Le champ numérique* 11(1): 1, pp. 1–98. DOI: https://doi.org/10.16995/dscn.313

Submitted: 12 July 2018 **Accepted:** 14 January 2020 **Published:** 13 April 2021