History and the Future of Markup

Michael Piotrowski
Université de Lausanne, Section des sciences du langage et de l'information
`<michael.piotrowski@unil.ch>`

1. Introduction

The report of XML's death has been greatly exaggerated, but it is becoming obvious that the halcyon days are over. To be sure, XML has enjoyed tremendous success since its first publication as a W3C Recommendation in 1998. Nowadays there are few areas of computing where, in some way or another, XML does not play a role. There are probably hundreds of specifications and standards built on XML, and dozens of related technologies, such as XSLT, XQuery, XPath, XML Schema, XLink, XPointer, XForms, etc. The W3C press release on the occasion of the tenth anniversary of XML quoted Tim Bray, one of the editors of the XML Recommendation, as saying, "[t]here is essentially no computer in the world, desk-top, hand-held, or back-room, that doesn't process XML sometimes."[1] This is certainly still true today: at the height of the hype, XML found its way into so many applications, from configuration files to office documents, it is unlikely to completely disappear anytime soon—even though it may become legacy technology.

Nevertheless, when it comes to formats for representing and exchanging structured data, JSON is now all the rage;[2] for narrative documents, Markdown enjoys a similar role. The W3C's XML working groups have all been closed, and HTML development has been taken over by what is essentially an industry consortium that opposed the transition of HTML to XML. The primary syntax of HTML5 *looks* like SGML, but the specification explicitly states that HTML is *not* an SGML application: "While the HTML syntax described in this specification bears a close resemblance to SGML and XML, it is a separate language with its own parsing rules."[3]

Markdown and similar lightweight markup languages clearly offer writers a much more compact syntax for authoring simple documents, but even slightly more complex documents require extensions. Consequently, people have defined a large number of mutually incompatible extensions for different purposes. Pandoc[4] does an amazing job at integrating many of them into a useful whole; one is reminded of this 1989 speculation about the future:

[1] W3C XML is Ten! [http://www.w3.org/2008/xml10/xml10-pressrelease]
[2] Sinclair Target, "The Rise and Rise of JSON [https://twobithistory.org/2017/09/21/the-rise-and-rise-of-json.html]"
[3] HTML Standard [https://html.spec.whatwg.org/#parsing]

A new generation of software products may change this: perhaps the most desirable result would be that by virtue of the markup minimization capability, together with smart editors, authors will be using SGML without knowing about it, whilst publishers reap the benefits. [1]

Except for the SGML part, of course: none of this is formally standardized.[5]

JSON and Markdown are in some respects certainly more convenient than XML, but hardly "better" in an absolute sense, in particular not from a computer science perspective. By defining an SGML DTD for HTML 5.1, Marcus Reichardt has demonstrated that, "while nominally not based on SGML, owing to HTML requiring legacy compatibility, HTML5 hasn't striven far from its SGML roots, containing numerous characteristics traceable to SGML idiosyncrasies." [20]. This, as well as the bitter conflicts between the W3C and WHATWG,[6] also suggests that the dissociation of HTML5 from XML and SGML is not due to technical requirements. It rather goes to show that the development of technology is not solely determined by the technical superiority of one system over another, but that it is to a large extent also driven by cultural forces, fads, and fashions. As technology is created and used by humans, it is a cultural artifact.

On the one hand, it is thus more or less unavoidable that preferences change for apparently "no good reason," or that small practical advantages in one area are used to justify giving up important benefits in other areas. On the other hand, given the investments made into the creation of a complex ecosystem such as that of XML, it would be a shame to simply throw it all away. This obviously applies to business investments, but what is much more important is the *intellectual* investment, the experiences and insights gained in the process.

2. Why we Need a History of Markup

This is why we need history of technology, in this case: a history of markup technology. If we want to advance the field rather than reinvent the wheel, we need to know by which ways we arrived at the point where we are now—including the roads *not* taken. Software, including markup languages, file formats, etc., are a very peculiar class of artifacts: as they are not governed by the laws of physics, designers enjoy, for better or worse, almost unlimited flexibility.

[4] Pandoc: a universal document converter [https://pandoc.org/]

[5] The CommonMark [https://commonmark.org/] initiative is working on a "standard, unambiguous syntax specification for Markdown, along with a suite of comprehensive tests to validate Markdown implementations against this specification."

[6] The conflict is even evident at many points in the HTML Standard [https://html.spec.whatwg.org/multipage/parsing.html#parsing]; for example, it—correctly—notes that "few (if any) web browsers ever implemented true SGML parsing for HTML documents" and that "the only user agents to strictly handle HTML as an SGML application have historically been validators." Claiming that this "has wasted decades of productivity" and that HTML5 "thus returns to a non-SGML basis" however, can only be interpreted as a dig at the W3C.

There are of course papers that take a historical perspective on markup and related technologies. For example, noting that "[d]ocument preparation has been an increasingly important application of computers for over twenty-five years," Furuta set out in 1992 to "identify those projects that have been especially influential on the thinking of the community of researchers who have investigated these systems" [5]. However, this overview (which covers publications up to 1988) was not intended as a *history* of document preparation, i.e., it does not link developments and suggest causalities. An actual history of markup would be a topic for at least one PhD thesis. The goal of this paper is thus merely to encourage a reflection on the history of markup, using SGML and XML as an example.

3. The Historical Trajectory of SGML and XML

As is well known, XML is an evolution of SGML, or, in the words of Egyedi and Loeffen [4], XML was "grafted" onto SGML. It thus has a clearly identifiable direct historical predecessor. The history of SGML has been documented several times by its "father," Charles Goldfarb, for example in Appendix A of The SGML Handbook, "A Brief History of the Development of SGML" [8]. SGML is an evolution of GML, a set of macros for IBM's SCRIPT formatter (itself modeled on Jerry Saltzer's RUNOFF) inspired by the idea of *generic coding*, which emerged in the late 1960s. Generic coding describes the idea of marking up text elements for their function (e.g., "heading") rather than their appearance (e.g., "Helvetica Bold 14/16, centered"). Generic coding thus introduced an abstraction and advanced the separation of content and form. Invented around the same time, Stanley Rice's *text format models* [21] can be considered the counterpart of generic coding in that it permits designers to systematically map these abstract structures to concrete formatting.[7] Taking these ideas together, one could thus mark up text as "heading" and then independently specify that headings are to be mapped to the model "LDa" (14-point sans serif bold) for one application or "LBd" (12-point text bold italic) for another—or the information "heading" could be used for the purpose of information retrieval. It is not hard to see that these ideas were very appealing for applications such as legal publishing, where highly structured texts are to be formatted in various ways for print and ideally also made available in electronic form, and thus also for IBM [7].

Goldfarb then went on to lead the development and standardization of GML into SGML, published as an international standard in 1986 [10]. In "The Roots of SGML—A Personal Recollection,"[8] he writes:

[7] In fact, Goldfarb has stated that "Stanley Rice, then a New York book designer and now a California publishing consultant, provided my original inspiration for GML." [7]

[8] Charles F. Goldfarb, "The Roots of SGML—A Personal Recollection [http://www.sgmlsource.com/history/roots.htm]"

> *After the completion of GML, I continued my research on document structures, creating additional concepts, such as short references, link processes, and concurrent document types, that were not part of GML. By far the most important of these was the concept of a validating parser that could read a document type definition and check the accuracy of markup, without going to the expense of actually processing a document. At that point SGML was born – although it still had a lot of growing up to do.*

The history of XML obviously does not begin with the publication of the W3C XML 1.0 Recommendation in 1998, nor with the creation of the SGML Editorial Review Board (ERB) by the W3C, which developed it, but the Web rather represented an incentive to revisit earlier proposals for simplifying SGML, such as Sperberg-McQueen's "Poor-Folks SGML"[9] or Bos's "SGML-Lite"[10]—or, as DeRose put it, "XML stands within a long tradition of SGML simplification efforts" [3]. From today's perspective, this historical development appears completely logical, as it follows a familiar narrative: from humble beginnings to great success. However, it is important to recognize that this well-known narrative is just *one* of many possible narratives. It mentions neither alternative approaches nor criticism, nor failures. The Office Document Architecture (ODA, ISO 8613) [12] is all but forgotten today, but it is one example of a quite different contemporaneous approach to more or less the same goals as SGML.[11] Criticism of the very idea of embedded markup is far from new either; for example, Raymond et al. claimed in 1993 that since "the formal properties of document management systems should be based on mathematical models, markup is unlikely to provide a satisfactory basis for document management systems" [19]. Robin Cover has called this report "a reminder that SGML has had significant intelligent detractors from the beginning."[12] In fact, many of these lines of criticism continue until today.

4. Some Observations on SGML

At this point we would like to point out some historically interesting observations that—while far from being obscure—seem to be less often discussed than, for

[9] Michael Sperberg-McQueen, "PSGML: Poor-Folks SGML: A Subset of SGML for Use in Distributed Applications [http://www.tei-c.org/Vault/ED/edw36.gml]," Document TEI ED W 36, October 8, 1992.

[10] Bert Bos, "'SGML-Lite' – an easy to parse subset of SGML [https://www.w3.org/People/Bos/Stylesheets/SGML-Lite.html]," July 4, 1995.

[11] The Wikipedia article Open Document Architecture [https://en.wikipedia.org/wiki/Open_Document_Architecture] (*Open Document Architecture* is the name used by ITU for their version of the otherwise identical standard) states: "It would be improper to call ODA anything but a failure, but its spirit clearly influenced latter-day document formats that were successful in gaining support from many document software developers and users. These include the already-mentioned HTML and CSS as well as XML and XSL leading up to OpenDocument and Office Open XML." Given the cleavage between SGML and ODA approaches and communities [15], we find this statement rather dubious without further support.

[12] http://xml.coverpages.org/markup-recon.html

example, the verbosity of the Concrete Reference Syntax (which is the only syntax for XML) or the problem of overlapping markup.

SGML is an extremely complex standard, and, as DeRose has remarked, the "list of SGML complexities that do not add substantial value is quite long" [3]. Some of these complexites are easy to explain historically. One example is markup minimization, which does not only include facilities for omitting start and end tags, but also *data tags*, text that functions both as content *and* as markup:[13] these features were motivated by the desire to minimize the amount of typing necessary when creating SGML documents with a simple text editor. Another example is the *character set description* in the SGML declaration, necessitated by the diversity of character sets and encodings in use at the time.

The reasons for other complexities are, however, less clear. For example, despite SGML's roots in a commercial product and extensive experience with it, many aspects necessary for interoperable implementations were left undefined, such as the resolution of public identifiers. Similarly, SGML does hardly say anything about documents may be processed apart from validation, in particular how they could be formatted for display or transformed for an information retrieval system. Macleod et al. criticized this in 1992 as follows:

> *SGML is a passive standard. That is, it provides mechanisms through which descriptive markup to be applied to documents but says nothing about how these documents are to be processed. The SGML standard refers frequently to the "application" but includes no clean mechanism for attaching applications to SGML parsers. [14]*

In fact, formatting seems to have been hardly a concern, as there is little in the standard related to formatting SGML documents or for interfacing with a formatter, and the facilities that are available—namely, link process definitions (LPD)—are not only weak, but also extremly complex, in particular in relation to what one can accomplish with them. In the commentary to Appendix D of the standard, entitled "LINK in a Nutshell," Goldfarb himself notes that "the problem is not so much putting LINK in a nutshell as keeping it there" [8]. This is well known—also because most of these features were removed from XML—but the question is why so much effort was expanded that quickly turned out to be of little use.

Another observation is that the SGML standard is strangely detached from computer science terminology and concepts that were already well-established at the time when the work on it started (between 1978 and 1980). Some of this is related to terminology, such as the use of the term *parser*: Barron noted in 1989 that the term *parser* is "firmly established in the SGML community" (in fact, it is

[13] In the SGML Handbook, Goldfarb calls data tags "to some extent an accident of history" and despite having been largely supplanted by short references, "still quite complex for both user and implementer" [8].

defined and used by the standard), whereas "a computer scientist would recognise the SGML processor as a parser generator or compiler-compiler which takes a formal specification of a language (the DTD) and generates a parser for that language, which in turn is used to process the user's document" [1].

Some problems are more serious; DeRose points out that "since SGML was developed with a publishing viewpoint largely divorced from computer science and formal language theory, it made some choices that led to bizarre consequences for implementers and users." [3] Kaelbling noted in 1990:

> *Since SGML is a language intended for computer-based systems, it is reasonable (in the absence of convincing argument to the contrary) that it should follow established conventions within the realm of computer science. By following accepted methods of notation and structural configuration, languages can be processed by existing tools following well-understood theories and techniques, thus offering considerable savings to development efforts. These savings are realized by automatically performing tedious, error-prone calculations correctly and by checking and clarifying the formal descriptions of the languages. [13]*

In their 1993 review of SGML, Nordin et al. made a similar statement:

> *From a software engineering viewpoint, it would make sense to use whatever tools are available to build SGML-based software. It is therefore of some interest to make sure that a standard does not inadvertently complicate the product development process.*
>
> *As specified, SGML does not cleanly map to either LL(1) or LALR(1)-type grammars. This has made it difficult to build SGML applications as the commonly used implementation tools have been difficult to apply. [16]*

Another formulation of this criticism: "It is not possible to construct a conforming SGML parser using well known compiler construction tools such a Lex and Yacc, since the language is context sensitive." [18] The demand that the "specification of a standard should reflect the state of the art" and that to this end, "the grammar specifying SGML should be rewritten to allow for automatic processing by common tools and techniques" [13]] seems altogether reasonable. Nordin et al. [16]] refer to Kaelbling [13] as well as further authors to underline this point; they also point to dissenting opinions (including Goldfarb's). This is another case of a problem that is well known, but again the question is: why? We suspect that the documents referenced by Nordin may be difficult to obtain now, but they could provide valuable insights on the rather striking fact that the editors of SGML, an international standard, were either unaware of or chose to ignore both research and proven practice in a relevant field.

A related observation is that even though it was clear to the editors of SGML that an SGML document describes a tree, SGML almost exclusively focused on the syntax. Goldfarb noted:

> *SGML can represent documents of arbitrary structure. It does so by modeling them as tree structures with additional connections between the nodes. This technique works well in practice because most conventional documents ar in fact tree structures, and because tree structures can easily be flattened out for representation as character sequences.*
>
> *Except for the terminal nodes, which are "data", each node in an SGML document tree is the root of a subtree, called an "element". The descendants of a node are the "content" of that element. [8]*

Despite that fact that parsing SGML documents shares many commonalities with parsing programming language code—and this is also mentioned in the standard —the parallels between the tree represented by an SGML document and the abstract syntax tree (AST) produced by a parser for a programming language seem not to have been apparent for the longest time. Considering an SGML or XML document as a serialization of a tree that exists independently of the concrete serialization (rather than the other way round) is a very powerful notion, especially in conjunction with a standard API to manipulate the tree.

In hindsight this appears obvious, but historically it seems to be a realization that was apparently not obvious at all. This does not mean that nobody had thought of it. For example, Furuta and Stotts already noted in 1988 that one of the most important problems to be solved in document processing is "to determine how composite document objects may be converted from one structure to another" [6]] and presented a system for transforming document trees based on H-graphs [17], explictly mentioning SGML as a possible application. However, only with the Document Object Model (DOM) and XPath the idea that an XML document describes a tree that is *independent* of its serialization, and on which programs can operate, took hold and eventually became explicit.[14] It is this notion that we think is the real foundation of today's XML ecosystem.[15]

5. The Future of Markup

Price noted in 1998 that the "historical origins of SGML as a technique for adding marks to texts has left a legacy of complexities and difficulties which hinder its wide acceptance." [18] This was proven true by XML: it is fascinating how quickly it was adopted and how an extremely rich ecosystem developed once many of these complexities had been discarded.

The early adoption by US Department of Defense and other government agencies (even before the publication of the final standard) was probably the decisive

[14] Conceptually this notion obviously already existed in DSSSL [11], but it remains for the most part implicit. For example, the standard talks about "transforming one or more SGML documents into zero or more other SGML documents" (page 9), i.e., it is the serialized documents that are considered primary.

[15] And which allows XQuery, for example, to process JSON just like XML.

for SGML to survive despite all of its problems. Looking back, it seems that a closer link of SGML to computer science research would have made it much easier to create tools and thus promoted a wider adoption. It may also have permitted people to realize earlier that the abstract tree structure is more important than the concrete syntax;[16] this would have greatly reduced the importance attributed to syntax.

It is also interesting to see that markup minimization—arguably the most syntax-focused feature of SGML there is—was a central concern. Due to the problems it created, it was completely rejected by XML—the XML Recommendation famously states as design goal 10: "Terseness in XML markup is of minimal importance."[17] Apart from the necessity to abandon markup minimization to make DTD-less parsing possible, one could say that the focus on markup syntax had become obsolete by the time it was realized that it is the abstract tree that is important. However, it probably also caused the backlash that we can now observe in the rise of JSON, Markdown, and similar formats that are essentially minimized HTML, as well as the reintroduction of minimization into HTML5.

Already in 1994, a critic pointed out that "SGML relies on technology from the 1970s, when almost all computers were mainframes, almost all data was textual, and sharing data—much less applications—between hardware platforms was almost unheard of" [9]; Price noted in 1998 that "SGML 86 reflects thinking from the 1960's and 1970's. Texts were not supposed to vary while being read" [18]. In the last 20 years both the types of content and users' interaction with it have significantly changed: at least on the Web the assumption that a document is static data that is parsed and rendered in what essentially amounts to batch mode is no longer true. This becomes evident in the HTML5 specification:[18]

> *This specification defines an abstract language for describing documents and applications, and some APIs for interacting with in-memory representations of resources that use this language.*
>
> *The in-memory representation is known as "DOM HTML", or "the DOM" for short.*
>
> *There are various concrete syntaxes that can be used to transmit resources that use this abstract language, two of which are defined in this specification.*

[16] In this context, Eliot Kimbers reflection in "Monastic SGML: 20 Years On [http://drmacros-xml-rants.blogspot.com/2013/08/monastic-sgml-20-years-on.html]" are interesting: "As I further developed my understanding of abstractions of data as distinct from their syntactic representations, I realized that the syntax to a large degree doesn't matter, and that our concerns were somewhat unwarranted because once you parse the SGML initially, you have a normalized abstract representation that largely transcends the syntax. If you can then store and manage the content in terms of the abstraction, the original syntax doesn't matter too much."

[17] Extensible Markup Language (XML) 1.0 (Fifth Edition) [https://www.w3.org/TR/REC-xml/#sec-origin-goals]

[18] HTML Standard [https://html.spec.whatwg.org/multipage/introduction.html#html-vs-xhtml]

These three sentences alone reflect significant conceptual differences to SGML, that are not evident from the seemingly conservative syntax. These include the formulation "documents and applications," the central role the DOM and the APIs to manipulate it, and the decoupling of the DOM from a concrete syntax. When XML was introduced, a frequently mentioned advantage over HTML was the possibility for users to create their own elements—the semantics of which would have to be defined by some application. HTML5 introduces the notion of *custom elements*,[19] which superficially seems equivalent. In fact, however, both the elements *and* their behavior are specified programmatically (i.e., in JavaScript) through the DOM API by subclassing `HTMLElement` (or a subclass thereof). This is a very different approach: first, because the element is not defined on the markup level but on that of the DOM, and second, because it also directly associates semantics with the element.

While HTML documents have for quite some time been a mix of HTML and JavaScript code operating on the DOM, custom elements represent a significant conceptual shift: HTML documents now have definitely become programs, with markup just serving as "DOM literal" or a kind of "here-document" for JavaScript. Is this the future of markup? In any case, this is the direction HTML is taking.

6. Conclusion

XML has come a long way. Its development and that of its ecosystem into what we have today is the result of over 50 years of history of document processing. Some choices were made consciously, others less so, and some features can only be described as historical accidents. We must look back in order to understand what lies ahead of us; taking a historical perspective can help to uncover hidden assumptions, implicit notions, critical decisions, misunderstandings, and so on, which still shape our understanding today. For Despite their similar appearances, HTML5 is conceptually quite different from SGML and XML. On the other hand, despite its different appearance, Markdown is very close to the traditional processing model of SGML. Its growing popularity in more and more domains requires more and more extensions, but it lacks a well-defined extension mechanism. The CommonMark[20] initiative aims to define and standardize a single common vocabulary, whereas one of the fundamental assumptions of SGML and XML is that this is not possible.

Sperberg-McQueen said in 1992 that "part of its [SGML's] accomplishment is that by solving one set of problems, it has exposed a whole new set of problems."[21] This is particularly true in a historical perspective. The point of this paper is to encourage reflection on and discussion of the history of markup tech-

[19] HTML Standard [https://html.spec.whatwg.org/multipage/custom-elements.html#custom-elements]
[20] https://commonmark.org/

nologies to advance the state of the art. In order to recognize opportunities and limitations of new technologies it is necessary to be able to compare it to previous technologies; at the same time, the design of new markup languages (understood in a wide sense) is, like that of programming languages, "always subtly affected by unconscious biases and by historical precedent" [2]; even approaches that aim to be "radically new" define themselves with respect to what has been there before. Those who cannot remember the past are condemned to repeat it.

References

[1] David Barron 1989. Why use SGML. *Electronic Publishing*. 2, 1, 3–24.

[2] Michael F. Cowlishaw 1984. The design of the REXX language. *IBM Systems Journal*. 23, 4, 326–335. doi:10.1147/sj.234.0326.

[3] Steven J. DeRose 1999. XML and the TEI. *Computers and the Humanities*. 33, 1–2, 11–30.

[4] Tineke M. Egyedi and Arjan Loeffen 2002. Succession in standardization: Grafting XML onto SGML. *Computer Standards & Interfaces*. 24, 4, 279–290. doi:10.1016/s0920-5489(02)00006-5.

[5] Richard Furuta 1992. Important papers in the history of document preparation systems: Basic sources. *Electronic Publishing*. 5, 1, 19–44.

[6] Richard Furuta and P. David Stotts 1988. Specifying structured document transformations. *Document Manipulation and Typography. Proceedings of the International Conference*. Cambridge University Press, Cambridge, 109–120.

[7] Charles F. Goldfarb 1997. SGML: The reason why and the first published hint. *Journal of the American Society for Information Science*. 48, 7, 656–661. doi:10.1002/(sici)1097-4571(199707)48:7%3C656::aid-asi13%3E3.0.co;2-t.

[8] Charles F. Goldfarb 1990. *The SGML handbook*. Oxford University Press, Oxford, UK.

[9] George F. Hayhoe 1994. Strategy or SNAFU? The virtues and vulnerabilities of SGML. *IPCC 94 proceedings. Scaling new heights in technical communication*. IEEE, New York, NY, USA, 378–379.

[10] International Organization for Standardization 1986. *ISO 8879:1986. Information processing — Text and office systems — Standard Generalized Markup Language (SGML)*. Geneva.

[21] Michael Sperberg-McQueen, "Back to the Frontiers and Edges. Closing Remarks at SGML '92: the quiet revolution [http://www.w3.org/People/cmsmcq/1992/edw31.html], October 29, 1992.

[11] International Organization for Standardization 1996. *ISO/IEC 10179:1996. Information technology — Processing languages — Document Style Semantics and Specification Language (DSSSL)*. Geneva.

[12] Vania Joloboff 1986. Trends and standards in document representation. *Text Processing and Document Manipulation. Proceedings of the International Conference*. British Computer Society; Cambridge University Press, Cambridge, 107–124.

[13] Michael Kaelbling 1990. On improving SGML. *Electronic Publishing*. 3, 2, 93–98.

[14] Ian A. Macleod, Brent Nordin, David T. Barnard, and Doug Hamilton 1992. A framework for developing SGML applications. *Proceedings of Electronic Publishing 1992 (EP 92)*. Cambridge University Press, Cambridge, 53–63.

[15] Charles K. Nicholas and Lawrence A. Welsch 1992. On the interchangeability of SGML and ODA. *Electronic Publishing*. 5, 3, 105–130.

[16] Brent Nordin, David T. Barnard, and Ian A. Macleod 1993. A review of the Standard Generalized Markup Language (SGML). *Computer Standards & Interfaces*. 15, 1, 5–19. doi:10.1016/0920-5489(93)90024-l.

[17] Terrence W. Pratt 1983. Formal specification of software using H-graph semantics. *Graph-grammars and their application to computer science*. H. Ehrig, M. Nagl, and G. Rozenberg, eds. Springer. 314–332.

[18] Roger Price 1998. Beyond SGML. *Proceedings of the Third ACM Conference on Digital Libraries (DL '98)*. ACM Press, New York, NY, USA, 172–181.

[19] Darrell Raymond, Frank Tompa, and Derrick Wood 1993. *Markup reconsidered*. Technical Report #356. Department of Computer Science, The University of Western Ontario.

[20] Marcus Reichardt 2017. The HTML 5.1 DTD. *Proceedings of XML Prague 2017*. University of Economics, Prague, 101–118.

[21] Stanley Rice 1978. *Book design: Text format models*. Bowker, New York, NY, USA.