Alma Mater Studiorum - Università di Bologna

DOTTORATO DI RICERCA IN
INFORMATICA
Ciclo XXVII

Settore Concorsuale di afferenza: 01/B1
Settore Scientifico disciplinare: INF01

Structural patterns for document engineering: from an empirical bottom-up analysis to an ontological theory

Presentata da: Francesco Poggi
fpoggi@cs.unibo.it

Coordinatore Dottorato:
Paolo Ciaccia

Relatore:
Paolo Ciancarini

Esame finale anno 2015

Abstract

This thesis aims at investigating a new approach to document analysis based on the idea of structural patterns in XML vocabularies. My work is founded on the belief that authors do naturally converge to a reasonable use of markup languages and that extreme, yet valid instances are rare and limited. Actual documents, therefore, may be used to derive classes of elements (patterns) persisting across documents and distilling the conceptualization of the documents and their components, and may give ground for automatic tools and services that rely on no background information (such as schemas) at all.

The central part of my work consists in introducing from the ground up a formal theory of eight structural patterns (with three sub-patterns) that are able to express the logical organization of any XML document, and verifying their identifiability in a number of different vocabularies. This model is characterized by and validated against three main dimensions: *terseness* (i.e. the ability to represent the structure of a document with a small number of objects and composition rules), *coverage* (i.e. the ability to capture any possible situation in any document) and *expressiveness* (i.e. the ability to make explicit the semantics of structures, relations and dependencies).

An algorithm for the automatic recognition of structural patterns is then pre-

sented, together with an evaluation of the results of a test performed on a set of more than 1100 documents from eight very different vocabularies. This language-independent analysis confirms the ability of patterns to capture and summarize the guidelines used by the authors in their everyday practice.

Finally, I present some systems that work directly on the pattern-based representation of documents. Since patterns can be extracted through automatic processing, these applications may operate on any document without any knowledge of its original schema. Moreover, the ability of these tools to cover very different situations and contexts (from the generation of presentation rules to information synthesis and extraction, from the exploration of the document content to the identification of document components) confirms the effectiveness of the model.

Acknowledgements

The work presented in this thesis has benefited from the input, support and suggestion of many people over the past years.

First of all, I wish to acknowledge my advisor, prof. Paolo Cianciarini, for his support and availability, and for the frank, valuable, constructive suggestions and critiques to my research activities.

I am extremely grateful to my tutor, prof. Fabio Vitali, who has involved me in his extraordinary research group and who has encouraged me and my work – it is really a pleasure working with you. Another thanks to the rest of my *commissione*, dott. Angelo Di Iorio and prof. Luciano Bononi, for their support and for having always been ready to discuss my Ph.D. topics.

Another big thanks goes to all my external referees – namely, dr. Michael Sperberg-McQueen, prof. Uwe M. Borghoff and prof. Oscar Corcho – for their precious and careful comments and advices.

I would like to express my deep gratitude to dr. Silvio Peroni, dr. Gioele Barabucci, dr. Andrea Nuzzolese and Roberto Amadini for having supported my work.

An infinite thanks to my family and Eugenia for their love, and for having uncon-

ditionally endured, supported and encouraged me in everything. This achievement is mine and yours as well.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

This thesis is positioned over two related research areas: markup languages and document engineering. The objective of this work is to present a language-independent model based on structural patterns that is able to capture the logical organization of *any* XML document, and express it in a simple and clear form. Moving off an analysis on the structure of documents, their basic constituents and composition rules, the thesis provides a definition of the model by introducing from the ground up a formal theory of eight structural patterns (with three sub-patterns), and verifies their identifiability on real world documents.

An algorithm for the automatic recognition of patterns is then presented, together with an evaluation performed on eight very different vocabularies for a total of more than 1100 documents. Finally, the thesis describes a set of systems built the basis of these ideas that help users in their everyday practice (e.g. to read and navigate document collections, perform complex analysis such as identifying and extracting document components, etc.) relying on no background information about document vocabularies, their intended meaning or schemas.

A central part of this thesis is devoted to the study of markup languages. XML schemas (be they DTDs, XSDs, Relax NG schemas, etc.) are the usual tools through which the regularity of a markup language is expressed. As with many other constraint languages, their purpose is both to delineate best practices, and to identify boundaries within which document instances can still be considered acceptable. The stricter the schema, the more these two tasks converge into one, but also, the more flexible the schema, the more difficult it can be to identify the middle ground of reasonableness within the wide variability of structures allowed by the grammar defined in it. Yet it is my belief that most authors do naturally converge to a reasonable use of a markup language and that extreme (although valid) instances are rare and limited. Actual documents (as opposed to their schemas), therefore, may give interesting insights into the expected characteristics of a vocabulary, and may give ground for automatic tools and services that are altogether independent of the schemas.

Is it possible to identify and exploit regularities in XML vocabularies regardless of the meaning of their terms and the availability of their schemas? Given the quantity of available XML documents in so many application domains, most of which are valid against well-known vocabularies, others compliant to niche or ad-hoc schemas, and still many not explicitly associated with any schema or associated with schemas that are not available anymore, is there any chance to be able to perform some useful operations on XML documents, without knowing any details of the rules with which these documents were composed, or the semantics associated with individual element names?

The research questions just enumerated guided my work to focus on the analysis and design of XML vocabularies regardless of their schemas. This perspective differs

from other works on XML validation since, instead of looking at the expressiveness of validation languages in defining element labels and imposing constraints on their use and positions, this thesis investigates document instances in order to derive classes of elements (patterns) persisting across documents and distilling the conceptualization of the document and its components.

The result of this analysis is a theory of structural patterns for XML documents that defines, from the ground up, a small set of eight fundamental patterns, plus three important subpatterns and some composition rules, that are sufficient to express what authors most frequently need (and actually use) in their documents.

I also try to answer a further question about document patterns, namely, to what extent authors actually use these patterns. Of course, documents are not naturally and fully compliant to the pattern model, and often schemas give authors a larger degree of freedom than structural patterns deem appropriate. Still, even with very general and open schemas, my tests show that authors do tend to adopt a simplified approach that is fundamentally pattern-based. For each schema, it is possible to identify reasonable pattern-compliant sub-schemas that most authors very often adhere to, and identify a small number of problematic elements and examples that are used in a non-pattern-based way.

The identification of these sub-schemas is not a goal I have, since compliance to it is often spontaneous and unrecognised by the very authors. Yet, *in practice* authors do tend to assign patterns to the elements of a vocabulary and use them accordingly, although with some exceptions I discuss in the thesis.

One straightforward application of the pattern theory is that it is possible to build useful applications automatically and without any previous knowledge about the vocabulary used, as I show in the last part of the thesis. For example, I present

the Document Viewer, a tool that provides an hypertext-like representation of documents and supports the user's navigation by generating index of terms, table of contents, and visual representations of the overall structure of documents. The information about the logical organization of documents summarized by structural patterns can also be exploited as the basis of further analysis, to grasp insight about the document itself. For example, I present the Document Component Extractor, another tool that supports the identification of higher level information such as document components (e.g. abstract, introduction, methods, problem statement, related work, etc.) in scholarly articles solely on the basis of the structural information provided by patterns.

Another important element in this thesis is the use of Semantic Web technologies. The first, broad motivation of this choice is that the direction in which I want to move my research is towards a semantic-enriched machine-readable document, where any information about the document itself is expressed in a clear and explicit manner, and can thus be easily retrieved, managed and used for further computations and analysis. For this purpose, the Web Ontology Language (OWL [79]) has been used to provide a formal definition of the theory of structural patterns. Then, I converted XML documents into EARMARK [41], a meta-markup language based on Semantic Web technologies, and exploited the OWL-DL reasoning capabilities in the engine that recognizes structural patterns. The transparent integration of these Semantic Web languages and technologies allowed the identification of meta-structures, as performed through the ontology, to be combined with other sources of information so as to validate the content at different levels of abstraction and to perform sophisticated queries and analysis, such as studying peculiarities of the documents.

Finally, another aspect worth noting is the use of visual analytics and information visualization techniques in the applications presented in the last part of the thesis. In particular, by revealing trends and patterns, and making explicit the semantics of relations, these techniques proved to be effective methods both for providing an overview of documents that supports reading and browsing, and for helping expert users to perform complex analysis on documents.

The structure of the dissertation is the following. Chapter 2 discusses related works and main issues in document engineering and markup languages. Chapter 3 describes the main elements and basic concepts that are the foundations of the pattern model, starting with some case studies. Chapter 4 provides a formal definition of the theory of patterns. Chapter 5 presents an algorithm for the automatic identification of structural patterns compliant with the theory, and an evaluation performed on an extensive set of vocabularies and documents. Chapter 6 describes some systems built using these ideas that work directly on pattern-based representation of documents. Final remarks and ideas for future works are in Chapter 7.

Chapter 2

Background

Technical, social and economic aspects have raised interest among researchers and professionals in the field of digital documents. Two research areas are particularly related to this dissertation: document engineering and markup languages. Document engineering investigates principles, tools and processes that improve our ability to create, manage, and maintain documents. Markup languages define objects, properties and rules to express information about raw text and approaches to text encoding.

In this chapter I discuss the most important issues in these areas, trying to outline which are the most relevant aspects of digital documents authors and designers have to deal with. In particular, I divide the analysis in two sections: first, in Section 2.1 I focus on document modeling, whose goal is understanding how a document can be represented in digital form, and second on (retrospective) document analysis, whose goal is understanding how the main components and relevant parts of that representation can be automatically identified and extracted from existing resources, as described in Section 2.2.

2.1 Document modeling

Although implicitly, authors face a lot of fundamental questions while writing a document: "Which logical structures do I need? How to organize the document content? How to highlight details and specific features?", and so on. When they write a digital document new issues need to be solved: "Which is the most suitable format? Which constructs should I use?", and in particular "Which markup language do I need?".

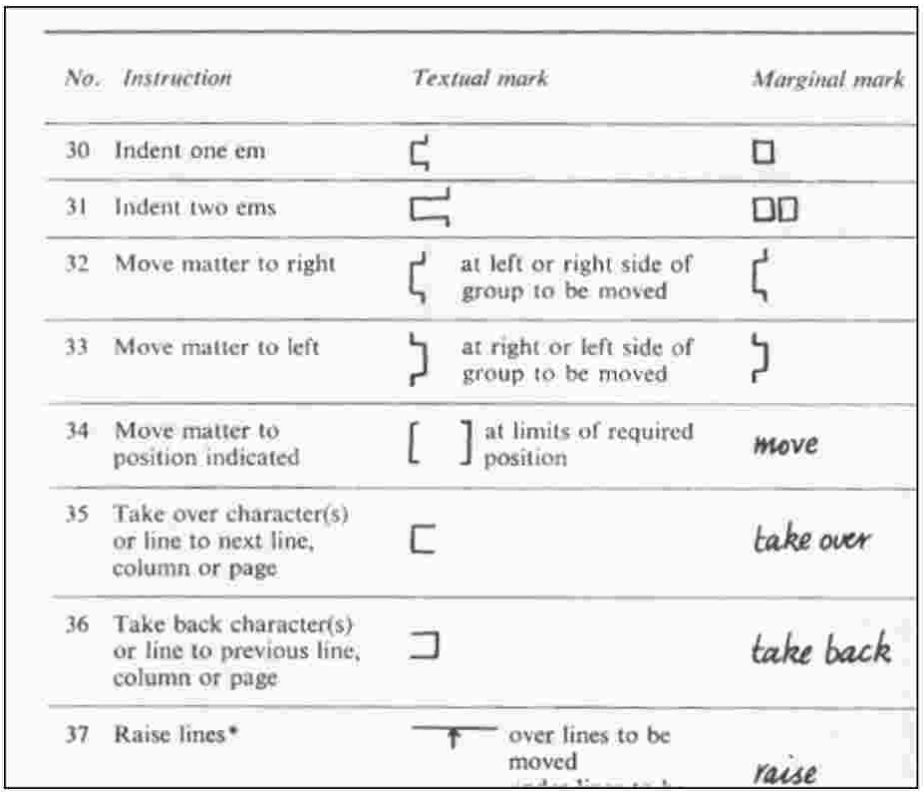
The following sections aims at answering these questions by providing a concise summary about the main concepts related to markup languages, focusing on the structural aspects and issues that concerns the overall organization of digital documents, which is the main topic of this thesis.

2.1.1 Markup and documents: an historical perspective

Before going into details, it is important to define the concept of markup. Historically, the word markup has been used to describe special marks or other annotations used to guide a compositor or typist on how a particular portion of text should be printed or laid out. For example, a straight underline to indicate italic, a wavy underline for boldface, specials symbols for fragments to be moved, aligned or printed in a particular font, and so forth. These editing (or proof-reading) marks have been used as a shorthand in copy-editing and proof-reading since the diffusion of Gutenberg's movable type mechanical printing technology since the 15th Century [3], and in a form similar to the modern ISO standard for proofreading [63][1] since the 17th

[1]The standard ISO 5776 specifies 16 symbols for text correction and proofreading. There are many other national standards for this purpose, such as the British standard BS-5261, the German standards DIN 16511 and 16549-1, the Italian UNI 5041:1996, etc.

Century [101]. A sample of proof marks and an example of their use in practice taken from a famous style manual [103] are shown in Fig. 2.1 and Fig. 2.2 on the next page respectively.



No.	*Instruction*	*Textual mark*	*Marginal mark*
30	Indent one em	⌐	□
31	Indent two ems	⌐	□□
32	Move matter to right	⌐ at left or right side of group to be moved	⌐
33	Move matter to left	⌐ at right or left side of group to be moved	⌐
34	Move matter to position indicated	[ ] at limits of required position	move
35	Take over character(s) or line to next line, column or page	⌐	take over
36	Take back character(s) or line to previous line, column or page	⌐	take back
37	Raise lines*	over lines to be moved	raise

Figure 2.1: Sample of proof mark guidelines from a style manual [103]

This interpretation of markup as diacritical signs[2] introduces us to a broader definition of the concept of markup. The TEI Guidelines [113] define markup, or (synonymously) encoding, as "any means of making explicit an interpretation of a text". Of course, whenever an author writes anything, he implicitly marks it up in this sense: for thousands of years, spaces have been used to indicate word boundaries, commas to indicate phrase boundaries, and periods to indicate sentence

[2]I use the term *diacr i tical* to refer not only to the characteristic of a symbol or a sign to indicate different phonetic values to the letters or words to which it refers to, but also to the ability of distinguishing words that are otherwise graphically identical. In this sense, the term diacritical denotes the function of a mark giving a special meaning to a part of text.

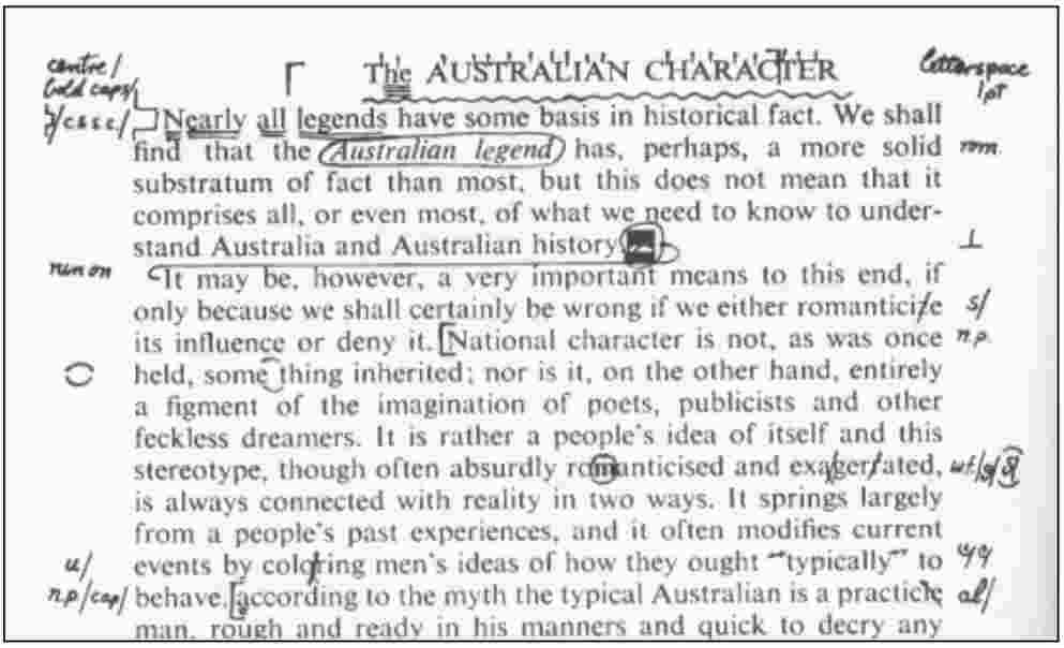


Figure 2.2: Example of proofing marks in practice

boundaries, and for hundreds of years page numbering[3] or margins have been used to structure the content.

At this point it is correct to wonder what is the relationship between text and markup and, in particular, whether the markup should be considered as part of the text. In [27], Coombs et al. clearly state that "markup is not part of the text, but it is used for saying *something* about the text": for example, no one will say aloud "comma" or "period" while reading a text, but will create appropriate paralinguistic behaviours (expressions, tones, pauses) in order to help listeners' understanding of the text.

Nevertheless, it is not difficult to show examples where a change in the markup implies a deep change in the nature of the text. For example, let's consider the two propositions "Let's eat, Grandmother!" and "Let's eat Grandmother!": the extent

[3]The use of page numbering was introduced after the transition from *volumen* to *codex*, even if incunabula and first manuscripts often lack page numbering. For instance, the Gutenberg Bible printed in the 1450s doesn't have page numbering.

of a small change in the markup (e.g. the existence/absence of a comma) is not just limited to the form of the text, but concerns also its interpretation, resulting in two different texts.

Markup is thus something that precedes the digitalization of texts: the examples above show that it is not only the unpleasant result of the computerization of printing, nor is it something that remain with us because of the information technology, but it is something independent and closely linked to texts.

2.1.2 Different objectives, different markup languages

Although markup existed before the advent of information technology, as briefly described in the previous section, there is no doubt that the development of text processing systems and their proliferation has led to new types of markup. When stored in electronic files, documents are indeed marked with special types of markup designed for processing by computer applications. The first step to understand the nature of digital documents consist in understanding the basic principles, characteristics and objectives of the languages they are written in. In the literature, many classifications were proposed, each useful for capturing some specific features.

In [27], Coombs et al. provide a classification of markup languages composed of six categories, which is still accepted nowadays[4]:

Punctuational: this type of markup consist of the use of a fixed set of signs to provide mainly syntactic information about the text. Punctuational rules are rather stable, familiar to authors and they are used frequently in documents, and for these reasons authors typically provide their own punctuational markup autonomously. However, there are significant problems and deficiencies in the use of punctuation:

[4]The examples used in this summary are taken from Coomb's original work.

- syntactic and structural uncertainty: e.g. comma, semicolon or period?

- stylistic variation: e.g. teachers and composition instructors often disagree on the use of commas after sentence-initial adverbial phrases;

- graphic uncertainty: e.g. often there is no agreement on the use of quotation marks, i.e. single opening and single closed, double opening and double closed or neutral/curly quotes;

- procedural ambiguity: e.g. the period is used to indicate abbreviations as well as sentence boundaries.

Authors that recognize these kind of issues often replace punctuational markup with other types of markup, such as referential or descriptive. For example, short quotations can be delimited by `&quot;` (a reference to the entity that represents the double quotation mark) or by `<q>` and `</q>` (a couple of opening and closing quotation tags): the use of one of these options allows authors to focus on the content of their text, and to postpone stylistic and graphic choices to a later time.

Presentational: for thousand of years authors have used presentational markup to make a clearer presentation of their texts. This kind of markup consists, for example, in vertical and horizontal spacing of portions of text, bullets for enumerating list items, page and section numbering, line-spacing to make reading easier, etc. Although authors have long inserted presentational markup in their writings by hand, now with the advent of electronic documents most prefer to have text processing systems automatically generate this kind of markup: page-numbering, for example, is a repetitive and error prone activity which is usually entrusted to automatic presentation systems.

Procedural: with the development of text-processing systems, presentational markup has been replaced by procedural markup. This type of markup consists in enriching the document with sequences of commands indicating how text should be formatted: these instructions are interpreted by an automatic system, which is responsible for their translation in concrete graphic effects that affect the final layout of the document. Famous examples of text formatters are nroff/troff [80] for Unix and TeX [69]. Procedural markup has often the drawback of being specific to a particular text formatter and, even worse, to a unique kind of device: an indentation of 50 pixels, for example, may be a suitable value for a desktop monitor, but too little for an high-resolution printer, or too much for a screen of a portable device.

Descriptive: the descriptive approach indicates to overlook the formatting and printing features and to focus on the structural role of each part of the document. Instead of specifying graphical effects such as alignment or spacing, the authors identify the role (e.g. title, section, paragraph, quote, etc.) of each text fragment. In order to do this, authors surround text fragments with special markers (called markup descriptors or tags) that indicate the beginning and the end of the portion of document with that particular role.

Referential: this kind of markup is used to refer to entities external to the document. In particular, referential markup is used to specify the meaning of the references, or indicate the graphic effect that should be used for their representation. I have already noted the use of referential markup for device-dependent punctuation (e.g., `&quot;` for a quotation mark). Another characteristic use is for abbreviations, such as `&acm;` for "Association for Computing Machinery". Referential markup might also refer to entities stored in a separate file or even on a different computing system.

Metamarkup: finally, metamarkup allows authors and designers to control the interpretation of declarative languages and to extend vocabularies in order to fit their needs.

With the advent of SGML and XML, descriptive markup languages soon achieved huge popularity. In [51] Goldfarb stressed two main benefits of descriptive approaches: generalization and rigorousness. Generalization concerns the ability of a document to be used in heterogeneous contexts, even very different from those for which they have been initially conceived and developed. The practical benefit is then that, once a document has been marked-up, all future processing can be implemented over that representation. Rigorousness means that the information about the content and the structure of a document are expressed in an unambiguous, clear and rigorous way, so that advanced and reliable applications can be actually built. Four other important features of descriptive languages outlined by Coombs et al. in [27] are maintainability, portability, minimazed cognitive demand and authoring enhancement. Although providing a complete list of references to the huge quantity of papers and books that have described the power, flexibility and applicability of descriptive languages is out of the scope of this work, I cannot omit citing the canonical references to Goldfarb's SGML Handbook [52], Sperberg-McQueen and Burnard Introductions to SGML [105], and XML [106].

Another important classification is that between prescriptive and descriptive DTDs, as described in [87]: a prescriptive DTD mandates a set of rules which must be followed by all documents and is primarily designed to create new material; a descriptive one describes structures and components that already exist, and is meant to create an electronic version of legacy texts. Extending this dichotomy to markup, prescriptive markup imposes constraints and rules about the organization, use, and

positioning of markup labels, while descriptive markup is mainly used for markup that simply defines some features of text fragments, without imposing any additional rule.

Another interesting class of markup has been described by Piez [84] as "exploratory/mimetic". This notion is used to describe all those languages (and, consequently, approaches to markup) that are not primarly meant to impose constraints about the organization of a document, but to simply describe document instances. The key aspect is the relation between an instance of document and its model: "the text to be marked up would be primary, the model merely a secondary and ex post facto expression of what the markup 'discovered' about the text" during the annotation process. For this reasons, Piez used the adjectives "mimetic" to indicate that a digital document aims at imitating its original source, and "exploratory" because it is adaptable to the characterisics of that source. Although the same author admitted that is difficult to justify a pure exploratory/mimetic language, he presented a fictional language called ProfML developed to be used in an exploratory way.

Renear [89] described other two dimensions that can be use to characterize markup languages, "domain" and "mood". The mood concerns the tone of a language, and it can be "indicative" (i.e. meant to describe something) or "imperative" (i.e. meant to impose something). The domain indicates whether a markup language (or part of it) refers to the logical organization or presentation of documents, and it can be classified as either "logical" or "renditional", respectively. In the logical domain, for example, an indicative element states that the tagged text fragment is a specific "object", independently from its markup; an imperative one states that the same fragment has to be modeled as that object.

Renear's imperative and indicative moods overlap with Piez's classification based

on time processing described in [84]. In fact, Piez's "retrospective" markup language seeks to represent something that already exists, as Renear's indicative moods, while the objective of a "prospective" language is to identify the constituents of documents, as Renear's imperative moods.

2.1.3 Format and content separation

One of the most accepted principles in designing markup languages is the separation of format from content. This principle is so well-accepted within the community that providing a complete list of citations is practically impossible: in fact, any decent book about SGML, XML and text encoding discusses that paradigm and the benefits resulting from its application.

The first paper I can't omit citing is the seminal work by Coombs et al. [27]: in this paper the authors, beside proposing a classification of the most important markup languages, outline the benefits of descriptive markup in terms of *maintainability*, *portability*, *cognitive demand* and *authoring enhancement*. Properly tagged files eliminate most of the mainteneance concerns: editing is simpler, files are protected from corruption and changes in the presentation does not affect the original file, since presentation is a separate activity that can be performed in a second phase. Moreover, document tagged with accurate and rigorous markup can be ported from one system to another over different platforms, since the actual meaning and logical organization of a document is captured by descriptive tags, and specific conversion can be straightforwardly performed by simple programs: different systems, different devices and different applications can display the same content simply by converting it on-the-fly. The process of marking-up documents itsef is simplified, since authors need only to select the most appropriate labels for content elements, and this re-

quires little more than the normal linguistic processing already necessary to perform element recognition. What authors called “descriptive markup” can be read as content/format separation: what really counts is the actual role and logical function of text objects, rather than their final rendering, formatting and processing.

Other two worth citing works are the introduction to SGML [105] and XML [106], in which Sperberg-McQueen and Burnard highlight some important benefits of content/format separation. In particular, they focused on the fact that ”the same document can readily be processed by many different pieces of software, each of which can apply different processing instructions to those parts of it which are considered relevant”. For example, a document with names of persons or places properly annotated might be used to create an index, or can be used as a source for data miners, etc. Similarly, a content analysis software might extract and analyze footnotes, a formatting program migth gather and collect them at the end of the document, and so on.

The diffusion of this approach has strengthened with the development of XML technologies, and has been consolidated with the standards proposed by the W3C. The use of CSS and XSLT recommended and encouraged by XML markup experts and the consortium, the increasing importance of multi-device and multi-platform issues, the proliferation of softwares and systems that embody that philosophy have made the principle of “separation between content and formatting” indissoluble from the concepts of content managment and advanced publishing. An almost infinite list of statements about the importance and benefits of XML content/formatting separation can be found in the literature: “XML helps us turn what is otherwise a stream of information into structured, manageable and meaningful data” by St. Laurent [109] ”, “the ability of XML is its ability to separate the user interface

from the data" by Pardi [81], "XML markup describes a document's structure and meaning. It does not describe the formatting of elements of the page" by Harold [57], etc.

Instead of further examining positive opinions, it may be useful to discuss some "opposite" positions that question a principle so widely accepted . An excellent critique of the idea is presented by Hillesund in [60]. The claim of the paper is that the doctrine of "one input – many output" supported by the XML community is basically wrong. On the contrary, the advent of new media, genres and formats (partially powered by XML) will lead publishers into a new and challenging state of "many outputs - many inputs". Hillesund's theory is based on two main points: *content/format interleaving* and *impossible reuse*. According to Hillesund, the separation of presentation from content is misleading when applied to publications such as books, because those two layers are so strictly interwoven and mutually dependent that there is no easy (and meaningful) way to separate them. The basic objection is that presentation is an irreplaceable part of a document that expresses some kind of semantic information, and thus affects not only the way a document is perceived, but also how it is comprehended by readers. For example, titles, introductions and chapters have both a semantic and typographic connotation, and authors actually use typographical elements when defining the logical structure of a document. The conclusion is that such a behaviour is so rooted in the history of typography and documents that XML cannot expect to separate elements that are intrinsically combined and have always been living together. The second point concerns the impossibility of reusing and merging fragments of content from different sources into an aggregate one. In fact, trying to rearrange the content will distort the logical order of elements, resulting in a document where the original information

is probably unclear, inadequate and too much complex. Such a reuse would be like "taking a pair of scissors, cutting up a tapestry weaving, rearranging it, and hoping to create a nice new weaving where all the threads are still connected". The conclusion is that, altough feasible from a technical point of view, there is no practical and meaningful way to manipulate fragments in a semi-automatic way: without a manual effort in readiting content, it is not possible to take a part of a book and reflow it into different layouts, for different purposes and different media, etc.

Walsh [119] wrote a point-to-point response to Hillesund's objections in the same journal. In this work, the author supported the principle of separation between content and structure, and presented some examples where the re-use of content on different media, with different layout and formatting produced very interesting results. The central point of Walsh's argumentation is that the ability to reuse the document content is dependent from how much the content is suitable to be extracted and reflowed: the core of the problem, in fact, is mostly editorial and cannot be solved by technical solutions. Altohough a bad document cannot be manipulated, reformulated and reformatted with perfect results, technologies that follow the principle of content/format separation provide a platform to build solutions, at least for those documents designed for that purpose.

Finally, other interesting and subtle discussions of the principle of separation between content and format have been presented by Liu in [73] and by Piez in [85].

2.1.4 Hierarchical models for digital documents: advantages and open issues

An important point for creating well engineered documents concerns the overall structure to use for organizing the textual content. The model to be used for digital

documents has been a topic of discussion since the very beginnings of work on markup languages.

An analytic and philosophical approach, the OHCO model, was discussed in the early 1980s by DeRose et al. in [34]. According to OHCO a text is an 'Ordered Hierarchy of Content Objects': a document is 'hierarchical' because elements nest inside one another like chinese boxes (a book contains chapters, which contain sections, which contain subsections, then paragraphs, then in-lines, down to the raw text); it is 'ordered' because there is a linear relationship among objects (for any two objects within a book one comes before the other), and it is made of plain units of information (content).

The adoption of the OHCO approach provides authors with many practical benefits, that can be divided in three main categories: composition assistance, production assistance and facilitation of alternate use of data. First of all, this approach let the authors focus on the logical organization of the document and the relations among its elements, rather than concentrating on other aspects as formatting, allowing to deal with the document at an appropriate level of abstraction. Writing, collaboration and alternate views on documents are all simplified since conceptual models are directly mapped into documents structures, relative relations are made explicit and different views of the same content can be easily created and updated. Moreover, since both the overall organization and dependencies among elements are explicit, advanced retrieval functions can be implemented, as well as functions of content composition and reflowing. The use of descriptive markup also simplifies the interchange and reuse of the document content between systems and applications.

Actually few counterproposals to OHCO were done (and they had a very low success), and OHCO suddenly became the most adopted model for designing markup

languages. The OHCO philosophical approach had been preceded by SGML, and was soon followed by XML, both offering further evidence of the flexibility and power of a hierarchical model.

But modeling documents as trees is not enough. For instance, when marking up text documents it might be necessary to represent features that do not fit into the tree structure conveyed by an XML document. There are many situations in which authors may need to annotate the same piece of text with different markup descriptors (e.g. when a page spans from the middle of one paragraph to the middle of another, or when speeches span multiple verses, etc.): in such cases, the markup descriptors sometimes nest correctly into a single tree-hierarchy, sometimes not.

In general, this issue may arises whenever an author wants to maintain two or more views of a document (e.g. metrical, syntactical, layout, etc.), and consequently multiple and incompatible hierarchies insists on the same textual content. This problem is referred to in the literature as the overlap problem.

After a first period in which the deficiencies of markup languages that concerns the overlap problem were underestimated [8][5] or even suppressed [34][6], the digital humanities community started to put an increasing effort in trying to define and develop solutions to this issue. The essence of the problem can be summarized as follows: "overlap can be presented by graphs that are very like trees, but in which nodes may have multiple parents. Overlap is multiple parentage" [108].

Since the document model of XML is inherently a tree, there is no simple way to cover such complex situations when handling multiple hierarchies. In order to

[5] In the first paper that deals with overlap in digital texts, in 1988 Barnard et al. argue that "SGML can successfully cope with the problem of maintaining multiple structural views", and that the solutions "can be made practical" by means of simple mechanisms, such as by exploiting the CONCUR feature of SGML [8].

[6] In [34], Renear et al. defend their OHCO thesis stating that "If you treat texts as ordered hierarchies of content objects many practical advantages follows, but not otherwise. Therefore texts are ordered hierarchies of content objects".

overcome these limitations, many different solutions have been proposed. For instance, the following list summarizes some of the most used XML-based techniques to manage overlapping situations:

- TEI-style milestones: this approach is to represent a vocabulary as primary by using a standard XML structure, and to use pairs of empty elements to mark the boundaries of elements that belong to secondary vocabularies. In order to make explicit the relation between corresponding opening and closing empty tags, a co-indexing mechanism may be implemented by means of special linking attributes [113][7];

- fragmentation: is another technique that prescribes breaking the elements belonging to secondary hierarchies into as many smaller fragments (also called partial elements) as needed to nest properly into the primary hierarchy. Also in this case overlapping elements are linked using special attributes (e.g. id-idref or next-previous pairs).

- stand-off markup: the key idea is to represent hierarchical and possibly incompatible structures separately from their actual content. In fact, the real content is present elsewhere, for example within the same document or in separate ones, and included by means of links implemented through a pointer mechanism such as XPointer [35]. In this way, it is possible to represent multiple conflicting structures as stratifications of different layers, at the cost of a overhead to manage and keep up-to-date the referenced content not directly

[7] It's worth noting that many slightly different types of milestones have been proposed: for example, another (more general) type of milestone consists in using milestone elements to mark the boundary between sections of a text, as indicated by changes in a standard reference system (e.g. the structure of pages in a standard codex). In those cases, each milestone element (except the first and the last) represents both the end of the previous feature and the beginning of the next one.

embedded within these structures.

- twin documents: overlapping hierarchies may also be encoded by using multiple documents that share the same textual content, but each one denoting its own tree structure, as described in [122].

Other worth citing solutions to overcome this limitation of XML are CONCUR [52] [107], JITT (Just In Time Trees) [44], MuLaX [59] [90] and Multi-colored trees [64]. Other approaches such as GODDAG (General Ordered Descendant Directed Acyclic Graph) [108], TexMECS [62] or LMNL [114] suggest to abandon XML and the benefits of its tree-based data model, and devise new formalisms and notations based on a more general and expressive abstract structure, such as a directed graph. The interested reader may find a complete description of these solutions in [33] or [74].

The discussion presented in this section about hierarchical models for digital documents is propaedeutic to introduce the ability of EARMARK [41], the meta-markup language I used to process documents in this work, to overcome some of the limitations of traditional markup languages to deal with complex document features: in Section 2.1.5, for example, I show how overlapping hierarchies can be easily expressed in EARMARK. Other characteristics such as the ability to add semantic annotations and to easily perform validity checks makes of EARMARK a suitable and convenient framework to perform the advanced operations on digital documents described in this thesis, such as those presented in Section 5.2 and Section 6.2.3. Moreover, this choice leaves open the possibility of planning new developments and applications for the techniques developed in this work: for example, it would be interesting to test the theory of pattern on documents with overlapping situations. This and other scenarios are briefly described in Chapter 7.

2.1.5 Semantic Web and markup languages

Beside enabling people to create data stores on the web, build vocabularies, and write rules for handling data, the development of Semantic Web technologies opens new perspectives for managing, linking and exchanging digital documents and their related information on the current web. Some recent works have investigated the possibility to use Semantic Web Technologies to provide a semantic enriched representation of XML documents, such as the DTD2OWL approach presented by Thuy et al. in [116], Vion-Dury's model to transpose XML/SGML/HTML documents into RDF triples described in [117], Bishof et al.'s transformation model from XML to RDF and back described in [13], etc. An updated summary of the research approaches aimed at providing interoperability and integration between the XML and Semantic Web worlds, together with a description of the resulting benefits can be found in [11].

In this section I focus on EARMARK (Extremely Annotational RDF Markup) [41], a different approach to meta-markup based on ontologies and Semantic Web technologies developed by my research group. The core of EARMARK is an OWL 2 DL [79] ontology[8] that defines document meta-markup. The basic idea is to model EARMARK documents as collections of addressable text fragments, and to associate such text content with OWL assertions that describe structural features as well as semantic properties of (parts of) that content. As a result, EARMARK allows multiple overlapping hierarchies where the textual content within the markup items belongs to some hierarchies but not to others.

The interesting point is that, in addition to solving some limitations of tradi-

[8]EARMARK Ontology: http://www.essepuntato.it/2008/12/earmark. The prefix *earmark* refers to entities defined in it, while the prefix *co* refers to entities – used in the EARMARK Ontology – defined in the old version of the Collections Ontology [24].

tional markup languages such as the problem of overlapping markup by allowing to express multiple hierarchies in a transparent way, EARMARK provides a convenient and complete framework to represent digital documents with complex features and perform advanced processing operations on them. For example, it can be used to generate validity constraints (including co-constraints currently unavailable in most validation languages) [42], to make explicit the semantics of markup [82], to annotate text or other markup documents [6], to keep track of changes in markup [83], and as interchange format to enable conversions between different kinds of XML vocabularies embedding overlap [7]. In particular, the ability to add semantic annotations and to easily perform validity checks, or its seamless integration with ontologies and reasoning tools, makes of EARMARK a suitable and convenient framework to perform the operations on digital documents described in this thesis, such as those presented in Section 5.2 and Section 6.2.3.

The whole ontological description of EARMARK is summarised in the Graffoo diagram[9] [45] shown in Fig. 2.3 on the following page. The core classes of the model describe three disjoint base concepts: *docuverses*, *ranges* and *markup items*.

The textual content of an EARMARK document is conceptually separated from its annotations, and is referred to through the *earmark:Docuverse* class. The individuals of this class represent the objects of discourse, i.e. all the containers of text from an EARMARK document. Any individual of the *earmark:Docuverse* class – commonly called a *docuverse* (lowercase to distinguish it from the class) – specifies its actual content through the property *earmark:hasContent*. There exist two different kinds of docuverses, those that specify all its content in form of a string (defined through the class *earmark:StringDocuverse*) and those that refer

[9]Graffoo is a graphical notation for OWL ontologies and it is available at http://www.essepuntato.it/graffoo.

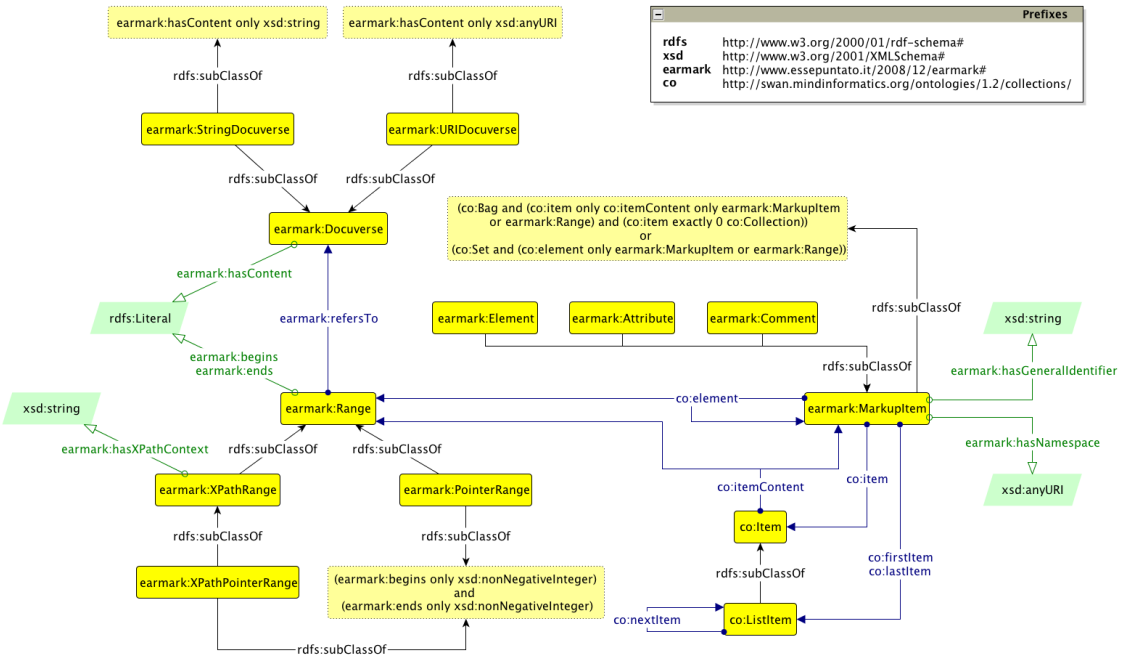


Figure 2.3: A Graffoo diagram summarising the EARMARK Ontology.

to a document containing the string to be marked up (defined through the class *earmark:URIDocuverse*).

The class *earmark:Range* is used to refer to text fragments lying between two locations of a docuverse. A *range*, i.e, an individual of the class *earmark:Range*, is defined by a starting and an ending location (any literal) of a specific docuverse through the functional properties *earmark:begins*, *eamark:ends* and *earmark:refersTo* respectively. There exist two main types of ranges: those (i.e., *earmark:PointerRange*) that refer to text lying between two non-negative integer locations that identify precise positions within a docuverse, and those (defined through the class *earmark:XPathPointerRange*) that refer to any text, obtained from a particular XPath context (specified through the property *earmark:hasXPathContext*) starting from a docuverse content, lying between two non-negative integer locations that identify precise positions.

The class *earmark:MarkupItem* is the superclass defining artefacts to be interpreted as markup such as elements (i.e., the class *earmark:Element*), attributes (i.e., the class *earmark:Attribute*) and comments (i.e., the class *earmark:Comment*). A *markupitem* individual is a collection[10] (*co:Set*, *co:Bag* and *co:List*, where the latter is a subclass of the second one and all of them are subclasses of *co:Collection*) of individuals belonging to the classes *earmark:MarkupItem* and *earmark:Range*. Through these collections it is possible:

- to define a markup item as a set of other markup items and ranges by using the property *co:element*;
- to define a markup item as a bag of items (defined by individuals belonging to the class *co:Item*), each of them containing a markup item or a range, by using the properties *c:item* and *co:itemContent* respectively;
- to define a markup item as a list of items (defined by individuals belonging to the class *co:ListItem*), each of them containing a markup item or a range, in which we can also specify a particular order among the items themselves by using the property *co:nextItem*.

A *markupitem* might also have a name, specified in the functional property *earmark:hasGeneralIdentifier*[11], and a namespace specified using the functional property *earmark:hasNamespace*.

In order to introduce some of the advanced features provided by the EARMARK model, I describe here how a document with three concurrent hierarchies can be

[10] In the following descriptions the prefix *co* to indicate entities taken from version 1.2 of the Collections Ontology [24], an imported ontology used for handling collections, available at http://swan.mindinformatics.org/ontologies/1.2/collections.owl.

[11] *General identifier* has been used to recall the SGML term *generic identifier,* that is the local name of the markup item, e.g., "p" for markup element " ... ".

represented in EARMARK, while other features of EARMARK will be presented when needed during the course of this dissertation. Let's consider the markup structures shown in Fig. 2.4.

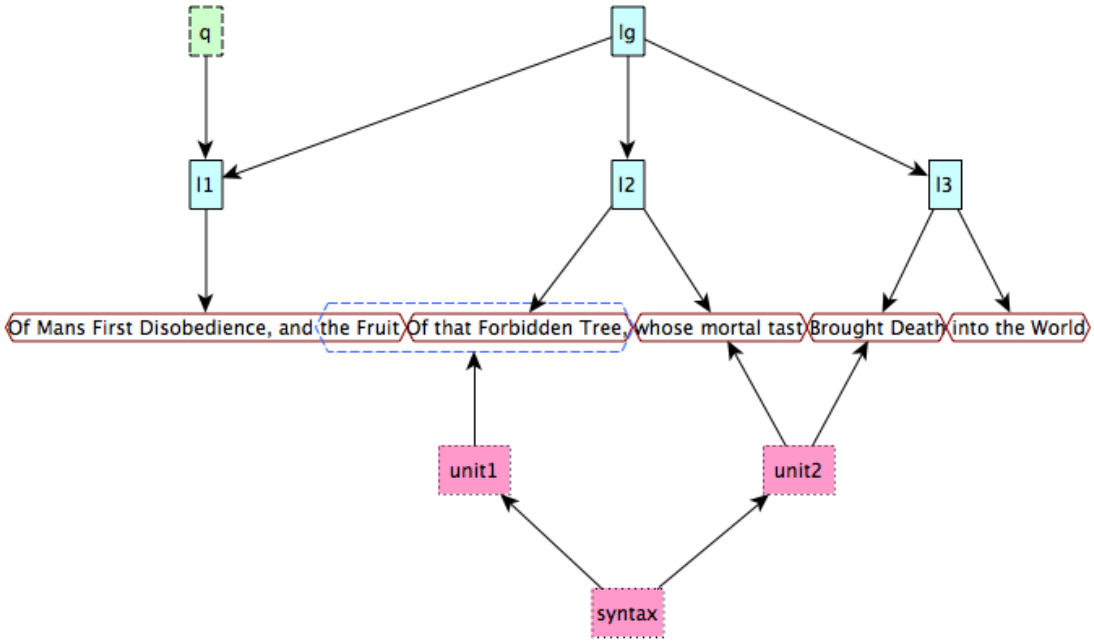


Figure 2.4: An example of three different markup hierarchies (light-blue rectangles with solid border, light-green rectangles with dashed border, and pink rectangles with dotted borders) involving six different ranges (the five empty rhomboids with solid red border and the one with blue dashed border).

In order to express the example in EARMARK, we start defining the whole textual content of the document – i.e., the first three lines of the *Paradise Lost* by John Milton – by creating an instance of the class *earmark:StringDocuverse*[12]:

```
@prefix : <http://www.essepuntato.it/2014/balisage/example/>

:doc a earmark:StringDocuverse ;

  earmark:hasContent
```

[12]This and all the following excerpts are defined in Turtle [86]. In the first excerpt, the content of the docuverse would have normalized whispace, as shown by character offsets in the following ranges, but that whitespace has been added to the Turtle example for the sake of clarity.

```
    "Of Mans First Disobedience, and the Fruit
    Of that Forbidden Tree, whose mortal tast
    Brought Death into the World" .
```

Then, we define all the six different ranges (as individuals of *earmark:PointerRange*) that are introduced in the figure, i.e.:

```
# The string 'Of Mans First Disobedience, and the Fruit'
:r1 a earmark:PointerRange ;
  earmark:refersTo :doc ;
  earmark:begins "0"^^xsd:nonNegativeInteger ;
  earmark:ends "41"^^xsd:nonNegativeInteger .

# The string 'the Fruit Of that Forbidden Tree,'
:r2 a earmark:PointerRange ;
  earmark:refersTo :doc ;
  earmark:begins "32"^^xsd:nonNegativeInteger ;
  earmark:ends "65"^^xsd:nonNegativeInteger .

# The string 'Of that Forbidden Tree,'
:r3 a earmark:PointerRange ;
  earmark:refersTo :doc ;
  earmark:begins "42"^^xsd:nonNegativeInteger ;
  earmark:ends "65"^^xsd:nonNegativeInteger .

...
```

Finally, we can build the three markup hierarchies shown in upon these ranges, as shown in the following excerpt:

```
:lg a earmark:MarkupItem , co:List ;
  earmark:hasGeneralIdentifier "lg" ;
  co:firstItem [
    a co:ListItem ;
    co:itemContent :l1 ;
  co:nextItem [
    a co:ListItem ;
    co:itemContent :l2 ;
  co:nextItem [
    a co:ListItem ;
    co:itemContent :l3 ] ] ] .

:q a earmark:MarkupItem , co:List ;
  earmark:hasGeneralIdentifier "q" ;
  co:firstItem [
    a co:ListItem ;
    co:itemContent :l1 ] .

:l1 a earmark:MarkupItem , co:List ;
  earmark:hasGeneralIdentifier "l" ;
  co:firstItem [
    a co:ListItem ;
    co:itemContent :r1 ] .
```

2.2 Document analysis

Authors of new digital documents take (or at least should take) into account all the principles discussed so far, in order to produce documents that can be easily stored, maintained and transmitted. This section is focused on the opposite process of document analysis, whose goal is understanding how the main components and relevant parts of existing documents can be automatically identified and extracted from legacy resources. There are in fact a lot of practical situations where huge document collections written in different formats and coming from very different domains and communities need to be collected, analyzed and re-structured.

For this reason, a lot of tools can be found in the literature about a posteriori analyses and re-structuring of heterogeneous digital documents and web pages. An exhaustive discussion is not in order here, but an overview of those techniques and, in particular, a discussion about the approaches and document models they adopt can be really useful for the purpose of this work.

2.2.1 Structural analysis of documents

Some literature has recently come out about the characterization and identification of structural components of text documents. For instance, Tannier *et al.* [112], starting from previous works by Lini *et al.* [72] and Colazzo *et al.* [26], describe an algorithm to assign each XML element in a document to one of three different categories: *hard tag*, *soft tag* and *jump tag*. *Hard tags* are elements that are commonly used to structure the document content in different blocks and usually "interrupt the linearity of a text": for instance, in the DocBook[13] vocabulary [121], they cor-

[13]DocBook is the format used in many examples and discussions in this and the following chapters of this dissertation. The main reasons at the basis of this choice are its popularity, and the availability of an extensive and clear documentation that can be easily consulted and examined

respond to, among others, *para*, *section*, *table* , etc. *Soft tags* are the elements that identify significant text fragments and are "transparent while reading the text": they are mostly inline elements carrying presentation rules (e.g., in DocBook, *emphasis*, *link*, *xref*, etc.). Finally, *jump tags* are elements that are logically "detached from the surrounding text" and that give access to related information – e.g., in DocBook, *footnote*, *comment*, *tip*, etc. Tannier *et al.* also introduce algorithms to assign XML elements to these categories by means of NLP tools. This classification is rather interesting, in that it provides a justification for the identification of the classes, but it is a little coarse for my purposes, ignoring empty elements and failing to distinguish higher level and lower level hard tags (i.e., those containing other tags but not text from those that never contain text). This classification partially overlaps with the pattern theory presented in Section 4.1: in particular, *soft tags* category is very close to the *Inline* pattern, the *jump tags* is similar to the *Popup* pattern, but their *hard tags* group comprises both *Block* and *Container* patterns.

Zou *et al.* [124] categorise HTML elements as belonging to two classes only: *inline* and *line-break* tags. Inline elements all those that do not provide horizontal breaks in the visualisation of documents – e.g., *em*, *a*, *strong* and *q*, while line-break elements are those that do so – e.g., *p*, *div*, *ul*, *table* and *blockquote*. Based on this categorisation and a Hidden Markov Model the authors try to identify the structural role (e.g., title, author, affiliation, abstract, etc.) of textual fragments of medical journal articles expressed as HTML pages. Higher-level structural roles (e.g., *div* elements used as section separators) are not discussed nor identified; similarly, out-of-flow elements (corresponding to jump tags in [112] and to the *Popup* pattern in the pattern theory) do not really exist as such in HTML and therefore are clearly

by the interested reader. Another important aspect is the availability of large document collections, that have been used to analyze and evaluate the effectiveness and accuracy of the theories, algorithms and tools presented in this work.

not identified.

Koh *et al.* [70] identify text fragments and images of documents that can act as their surrogates (where surrogates are defined as "information elements selected from a specific document, which can be used in place of the original document"). In particular, they address the issue of identifying *junk structures*, such as navigational elements of Web sites, advertisements, footers, etc., that usually do not carry the meaning of a document. Their approach is based on a pattern recognition algorithm that segments the XML elements of the document according to *tag patterns*, i.e., recurring hierarchies of nested elements that "contextualize the structured markup of text within a document". They find that junk structures are often described by similarly structured markup in different documents, and thus some tag patterns are crucial for their identification as junk within real HTML pages.

Similarly, Vitali *et al.* implemented a rule-based system for the analysis of regularities and structures within web pages [118]. The system seeks patterns in the HTML code of a page and labels the components of that page according to these patterns. One key aspect of the system is its extensibility. There is in fact a strong distinction between the rule engine and the actual patterns, which are declaratively expressed through XPath expressions in a custom XML vocabulary. Authors define an initial set of patterns to recognize, for instance: table cells, editable regions, navigational elements and annotated non-navigational text fragments.

Georg *et al.* [50] introduce an NLP approach to the automatic processing of medical texts such as clinical guidelines, in order to identify linguistic patterns that support the identification of the markup structure of documents. This approach justifies the development of a system for the automatic visualisation and presentation of unstructured documents. In a more recent paper [49] Georg *et al.* illustrate

an extension of such a work in which they introduce an improved version of their approach.

Finally, another worth citing reasearch area related to the structural analysis of documents is grammatical inference (also known as grammar induction). The general problem consits in extracting a grammar from sequential or structured data (e.g. strings, words, trees, etc.) by capturing regularities in their usage, and has been investigated by different communities such as machine learning, formal language theorists, pattern recognition, computational linguistics, etc., as described in [29]. In the context of digital documents, many algorithms and systems have been developed to capture the schematic information from SGML and XML document collections [67] [95] [21].

2.2.2 Analysis of document components

Several approaches have been proposed to extract logical components from paper-based documents. They can be divided in two main categories: bottom-up and top-down. Top-down algorithms start with the whole document and iteratively segment it into subcomponents, considering completed each segmentation step when a set of predefined properties are met. Bottom-up solutions start with single letters, then cluster them into words and paragraphs, then into graphical areas up to rebuild the whole document. For instance, in [65] authors presented a bottom-up approach for region identification, that exploits the connectivity and contiguity of graphical elements in order to extract text fragments, tables, images and drawings.

Very interesting results have also been achieved in automatic segmentation of web pages. Several works exploit visual features of pages in order to synthesise logical structures. For instance, in [19] a page layout is modelled as a graph of areas and

weighted strings elements for attributes and text. The logical tree is extracted from the nesting of elements and the weight of their subcomponent. Further information is collected by identifying subtrees and paths and by inferring elements names through a content-based analysis. Others, for instance [32] have proposed to use geometrical clues and spatial information to infer logical structures. They exploited a hierarchical representation of the screen coordinates of each page element in order to determine common areas in the page (headers, side menus, main content areas, footers, etc.) and their relations, and to infer their structural roles.

In [22] the authors propose a restructuring approach to derive properly nested XML documents from HTML pages by studying how HTML visual markup is related to the logical structure of a document. The authors model a document as a hierarchical structure of block-level and in-line-level objects, where objects of higher level of abstraction are described as combination of lower level objects. The meaning of a node is not directly associated with the object itself but it is related to the content and context of that node. Then, they propose a bottom-up process to restructure a DOM tree consisting of three steps: (1) analysing text in order to identify atomic units of content, (2) grouping those units in more complex structures and (3) polishing those structures by removing non-relevant or temporary information. The latter steps are described through a declarative language of composition and filtering rules.

Very good results in extracting structured information from documents have been achieved when focusing on specific domains. For instance, [104] and [20] targeted Web pages publishing news. They presented hybrid approaches – that take into account recurring tree structures as well as presentational features – and are able to automatically characterize titles, publication and authoring data and structured

content. [124] focused on medical content and in particular scientific journals. The authors classified content objects into in-line and block-level units and proposed an algorithm based on Hidden Markov Models that identify automatically the main components (metadata, abstract, title and body) of the input documents.

There are also several research works on the description of structural and rhetorical components of a document. For the rhetorical part, three main models exist for document segmentation: the *Ontology of Rhetorical Blocks* (*ORB*) [23], the *SALT Rhetorical Ontology* [53] [55] [56] and the *Medium-Grained structure* [31]. The former model offers a coarse-grained description (header, introduction, methods, claims, etc.) and the latter a medium-grained description (hypothesis, objects of study, direct representation of measurements, etc.) of the rhetorical components of a document.

Another interesting work is Zhang's taxonomy of functional units [123] (i.e. chunks of information with a distinct communicative function) based on Swales' genre model [111]. The main contribution consists in identifying the functions of the information units within four journal article components (i.e., introduction, methods, results, discussion), and their associations with information tasks performed by users. Encouraging experimental results showed the benefits of the use of functional units in supporting navigation, close reading and comprehension of journal articles.

Chapter 3

Structural Patterns for document engineering

Evaluating collections of XML documents without paying attention to the schema they were written in may give interesting insights about the expected characteristics of a markup language, as well as any regularity that may span vocabularies and languages, and that are more fundamental and frequent than plain content models. These regularities in the document structure (or *structural patterns*), as well as the benefits deriving from their identification, are the object of the investigation presented in this thesis.

My research group has already investigated the idea of using structural patterns for processing digital documents. In particular, in [37] Di Iorio et al. presented a minimal set of seven patterns that they used for integrating heterogeneous documents in a content management system. Taking their work as a starting point, I performed an analysis on real world documents and identified some deficiencies and limitations of the pattern model. For instance, I identified four very common situations (aka new patterns) that were not covered by the minimal set of patterns used to integrate documents. The result of this analysis is a new model for structural

patterns that extends and improves the previous one, and that is able to capture and express the logical organization of any document. Moreover, I identified two dimensions (i.e. content model and context) that are sufficient to characterize the most important structures in digital documents, and used these notions as the foundations of the formal definition of structural patterns presented in the next chapter.

This chapter is organized as follows. In Section 3.1 I describe the segmentation model based on structural patterns developed by my research group, and introduce the minimal set of patterns they used to capture the logical organization of documents. In Section 3.2 I present an analysis on real world documents, pointing out some limitations of the pattern model and proposing amendments and extensions. In Section 3.3 I discuss the results of the analysis and the design principles which form the basis of the complete theory of structural patterns presented in the next chapter.

3.1 A pattern-based segmentation model for descriptive documents

The idea of using patterns to produce high-quality and reusable assets is not new in the literature. Their inventor Alexander defined a pattern as "a three part rule, which expresses a relation between a certain context, a problem, and a solution" [1]. The basic idea is to capitalize previous experiences, in order to re-propose accepted solutions to recurring problems in similar contexts. The power of patterns lies in their ability to capture the core of a problem, and to suggest a guideline that experts can use to build their concrete solutions: indeed, as stated by the author, "each pattern describes a problem which occurs over and over again in our

environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over without ever doing it the same way twice" [2]. Moreover, patterns are the building blocks of what Alexander defines "a pattern language": as the elements of a language are units of information that are put together in order to build meaningful sentences, similarly patterns can be assembled to produce artifacts that meet complex needs and address hard problems. From this perspective, the benefits of adopting patterns is twofold: first, (by distilling best practices,) they are the actual solution to specific problem, second, they are used by experts as a powerful mean to communicate and investigate such solution.

Alexander was an architect and collected a set of solutions for building homes, workplaces, towns and cities into a pattern language that addresses a particular domain. Soon professionals and researchers understood the benefits of patterns in terms of effectiveness, flexibility and reusability, and started to apply this approach to different fields too. For example, Gamma et al. [46] provided a complete description of patterns for software development, and their book immediately became (and still is) a must-read resource for the community of software engineers and object-oriented experts. More recently, design patterns have been used in the domain of ontology development [47] [16], facilitating reuse and showing remarkable results in terms of quality improvement.

My research group has been investigating patterns for XML documents (e.g. [37] [28]) to understand how the structure of digital documents can be segmented into smaller components, which can be addressed independently and manipulated for different purposes. In order to express and capture the logical organization of heterogeneous documents, the group has defined a minimal model based on such

patterns, and used this model to project heterogeneous documents into an unified pattern-based representation. These ideas have been used to develop a group of systems that cover heterogeneous content management processes: from web editing to collaboration, from e-learning to professional printing.

This minimal set of patterns is the starting point of my work. Although a deep analysis of this model and its applications is out of the scope of this dissertations, in this section I provide an overview of the model and its applications, focusing on the elements and concepts that are useful for the purpose of this dissertation.

3.1.1 A document segmentation model: Pentaformat

The principle of separation between content and formatting discussed in Section 2.1.4, as well as the need of segmenting documents into subcomponents, is one of the most accepted (and widely discussed) principles among document engineers and markup experts. The analysis and segmentation of documents into subcomponents is so embedded and well-accepted by the community that providing a complete list of references is practically impossible (canonical references are [52], in particular Annexes A “Introduction to Generalized Markup”, B “Basic Concepts” and C “Additional Concepts”, [27] and [105]).

The most popular markup (meta-)languages (i.e. SGML, XML and HTML) and their related technologies (i.e. XPATH, XSLT, CSS, ...) have focused on the analysis and management of three main constituents:

- **Content:** the pieces of information in the document, the ”what is it” information, or the ”gray matter”
- **Structure:** the arrangement of the content, the ”where is it” information, or the ”skeletal matter”.

- **Presentation:** the formatting or rendering of both structure and content components; the "what does it look like" information; much of the time it doesn't matter except as it helps to identify components of the other two types.

Although this three level distinction provides many practical benefits, it is not difficult to find examples where such a classification is not enough, in particular considering the increasing importance of interaction and dynamic behaviour in some contexts, like the World Wide Web. For instance, interactive behaviours frequently violate any simple notion of a static marked-up document, since both content and presentation could depend on arbitrary computation and user interaction.

To address these issues, my research group has proposed an approach named Pentaformat [28], a segmentation model that can be used to capture the most relevant document constituents. The basic idea is to divide a document into subcomponents that express the same information (i.e. meaning, organization, behaviour, graphical impact, etc.) of the original document. The main benefit of this approach is that these components can then be processed, reformulated and reused in very different situations of the content management process. For example, the authors describe some systems built on top of these ideas that cover web editing, collaboration, e-learning and professional printing contexts.

As showed Fig. 3.1 on page 43, the Pentaformat is composed of five dimensions:

- **Content:** the plain information made of text and images (audio and video are not considered);

- **Structure:** the logical organization of the content. Structure is meant to indicate the role of each content element and their relations, in order to make a document interpretable and processable;

- **Presentation:** the set of visual and typographical features that maximizes the impact of the document on human readers. This layer is built on top of the structure, and is crucial to convey what is inherently expressed by structured content. It is one of the possible expressions of the original information, interpretable and appealing for human readers;

- **Behavior:** the set of dynamic actions of events on a document required to model the interaction between readers and digital documents;

- **Metadata:** the set of information about the document, which make individual resources easy to search, compare and manage within wide document collections.

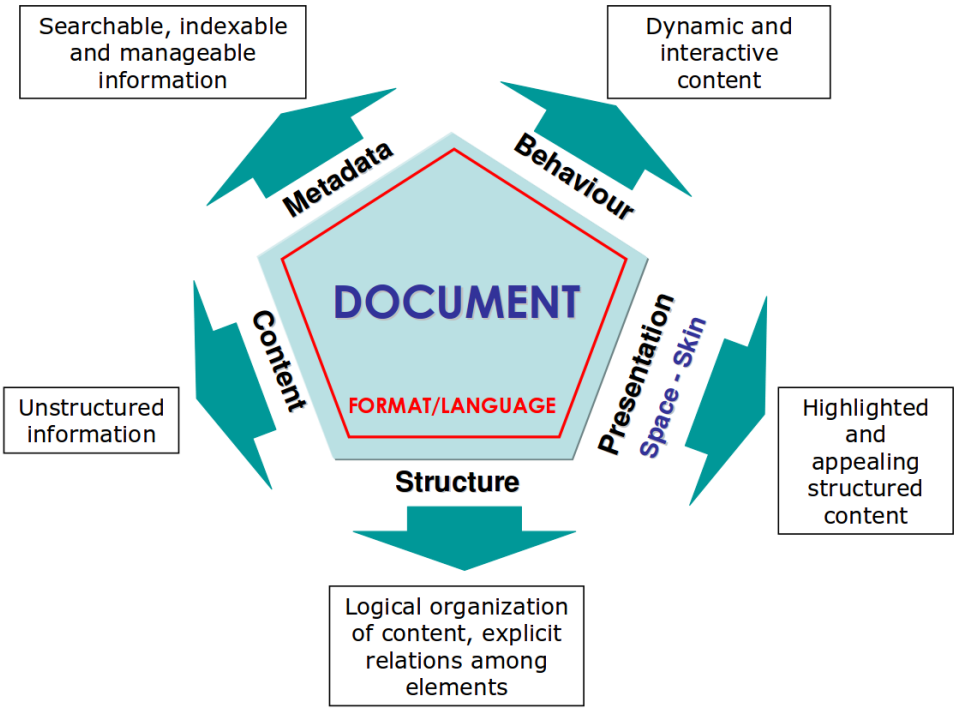


Figure 3.1: An overview of the Pentaformat Model that emphasizes the role of each constituent

3.1.2 Patterns for document substructures

Among all the dimensions of the Pentaformat, two are particularly important because they constitute the basic information written and organized by the authors. The correct addressing of the basic content and the logical organization of a document is not an easy task: in fact, markup experts and developers are required to deal with a wide variety of languages, formats and documents in their everyday practice. In order to reduce the complexity introduced by this heterogeneity, the authors of the Pentaformat present seven structural patterns, whose goal is to express in a clear and unambiguous form the building blocks shared by different vocabularies. The main characteristics of these patterns can be summarized as follows:

- **Marker:** an empty element, in case enriched with attributes, whose meaning primarily depends on its position within the context. A marker is not meant to provide characterization of the text content, but to identify special rules for a given position of the text. For example, they can be used to separate (sometimes visually) what comes before the marker from what follows (e.g. the elements `BR` and `HR` in HTML, or page delimiters).

- **Atom:** a unit of unstructured and not further divisible information. An atom contains only plain text and is meant to indicate a specific role or semantics for that information. Markers are used in text streams to capture the role of fragments, or as records that are assembled to build more complex structures;

- **Block** and **Inline:** both of these elements have a mixed content, and are used to organize textual content mixed with unordered and repeatable elements. They have the same role of carrying the text written by the author, with the difference that blocks can't contain other block elements recursively (i.e.

blocks use the the same content model of inlines, but are not listed in the allowed elements);

- **Record:** a set of heterogeneous, unordered, optional and non-repeatable elements. Records are first used to group simple units of information in more complex structures, or to organize data in hierarchical subsets where relations, dependencies and repetitions among elements are explicit;

- **Table:** a sequence of homogeneous elements. Tables are used to group similar objects into the same structure and, also, to represent repeating tabular data;

- **Container:** a set of heterogeneous, unordered, optional and repeatable elements. The name itself emphasizes the generality of this pattern, used to model all those circumstances where diversified objects are repeated and collected together.

Table 3.1: Patterns and Content-models in DTD syntax

Pattern	DTD syntax
Marker (`M`)	<!ELEMENT M EMPTY>
Atom (`A`)	<!ELEMENT A (#PCDATA)>
Block (`B`)	<!ELEMENT B (#PCDATA | M | A | I) $*$ >
Inline (`I`)	<!ELEMENT I (#PCDATA | M | A | I) $*$ >
Record (`R`)	<!ELEMENT R (Any-pattern-except-I?)>
Container (`C`)	< !ELEMENT C (Any-pattern-except-I)*>
Table (`T`)	< !ELEMENT T (A* | $B*$ | $R*$)>

Each pattern is characterized by a content model, which defines the element allowed within each pattern, as summarized in Table 3.1 on the preceding page. These composition rules express functional dependencies and relations between the document components (see Table 3.2), and make explicit the semantics of the structures within the overall organization of the document.

Table 3.2: Composition rules over patterns

	EMPTY	Text	Marker	Atom	Block	Inline	Record	Container	Table
Marker	X								
Atom		X							
Block		X	X	X		X			
Inline		X	X	X		X			
Record			X	X	X		X	X	X
Container			X	X	X		X	X	X
Table			X	X	X		X	X	X

Traditional pattern-based strategies are based on the principle of identifying the most useful solutions to recurring design problems, and reusing them. The authors of structural patterns suggest a different approach: instead of using patterns to guide designers in the creation of new and well-engineered resources from scratch (constructive model), their objective is to define a minimal model that is able to expresses the essential information about the organization of a document. In practice, the pattern model provides a uniform representation for heterogeneous documents, and is therefore used as the center of a conversion system for integrating, recombining and processing documents coming from different sources and formats. This simplification of the document meta-model has then been successfully exploited to support the content management process in very different situations: from web editing to collaboration, from e-learning to professional printing.

3.2 Structural patterns: an analysis

The objective of the minimal model presented in Section 3.1 is to simplify the document meta-model by identifying a minimal set of patterns that are able to express the most used and meaningful structures of digital documents. Considering

the simplicity and diffusion of the patterns proposed, the authors decided to describe them in a narrative style, as Alexander did with his patterns about architecture [2]. The lack of a rigorous definition produced by this approach is one of the main limitations of the structural patterns, and lays them open to criticism about their completeness, validity and applicability. Moreover, it is important to investigate their identifiability in real documents, and test their ability to capture all the main and more meaningful structures used by authors to organize the content of their documents.

Taking this minimal set of patterns as a starting point, my objective is to define a complete model of patterns that is able to express the logical organization of any XML document. In order to do so, in this section I describe an analysis I performed on real XML documents to check the limitations of the minimal set of seven patterns described in the previous section. The results of this analysis is presented in four subsections, one for each of the situations that are not well covered by the previous minimal set of patterns.

Another important objective of this analysis is to delineate a grammar of concepts, notions and relations that can be used to capture the main characteristics of the document structure, and to provide the foundations of a general model for structural patterns. In particular, I have identified two orthogonal and strongly related dimensions that can be used to characterize the semantics of the most relevant document meta-structures: *content model* and *context*. These two notions are the basis of the pattern theory, and are used to provide the formal definition of patterns presented in Section 4.1. There are also other relations that are important, such as the order of the elements in a content model, that can be used to provide a more specific characterization of some similar patterns, as described in Section 4.2.

3.2.1 Specialization of the Marker pattern: Milestone and Meta

According to the theory of structural patterns, a marker is an empty element whose meaning is strictly connected with its position within the document. Typical examples of marker are the element `HR` in HTML 4.01 [88], an horizontal rule whose role consists in separating visually what comes first from what comes next, or the element `IMG`, that is used to embed an image at the location of the element's definition.

Empty elements are also used for a different purpose: to insert metadata that asserts things about the document, but are disconnected from its actual content: in such cases, the position of the empty elements is often not important. In order to clarify these two different functions played by empty elements, let's consider the elements `XREF` and `COL` from the Balisage Tag Set[1], a small subset of the Docbook vocabulary [120] used by the Balisage Markup Conference[2]. The element `XREF` is the preferred method for inserting cross-references of any kind at specific locations, and is usually displayed as a link to the destination of the reference, and `IMAGEDATA` indicates an image resource along with information pertaining to its display.

```
<!ELEMENT  para  (#PCDATA | blockquote | citation | ... |
   xref)*>
<!ELEMENT  xref  EMPTY>
<!-- BalisageVol1-Brown01.xml -->
<para>There is also a note indicating that [<xref
```

[1] More information about the Balisage Tag Set, such as the DTD and its definition in [48] or [25], are available online at the address http://balisage.net/tagset.html. A clear and concise treatment of Docbook, the markup language from which the Balisage Tag Set stems from, can be found in [121].

[2] All the examples presented in this chapter are taken from papers presented at the Balisage Markup Conference. The complete proceedings of the conference are available online at the address http://balisage.net/Proceedings/index.html

```
   linkend="ISOTM"/>] did not explicitly define scope to
   include all subjects, therefore there may be some
   interoperability issues with older topic map instances
   [<xref linkend="TMDM"/>].</para>

<!ELEMENT mediaobject        (imageobject+, caption?)>
<!ELEMENT inlinemediaobject  (imageobject+)>
<!ELEMENT imageobject        (imagedata)>
<!ELEMENT imagedata          EMPTY>
<!-- BalisageVol8-Bruggemann-Klein01.xml -->
<mediaobject>
  <imageobject>
    <imagedata width="15cm" format="png"
      fileref="graphics/Bruggemann-Klein01-001.png"/>
  </imageobject>
</mediaobject>
```

In the first part of the example, cross-references are not meant to provide characterization of the text content, but to identify special rules for a given position of the text. The second part is an example of the opposite situation: the location of the element `IMAGEDATA` is not relevant, and what is important is the metadata information conveyed by mean of its attributes. For instance, the attribute `format` is used to declare the format of another part of the document, i.e. the image to which it refers.

The fact that the position of this element is not relevant is confirmed by the DTD: `IMAGEOBJECT` is a wrapper for `IMAGEDATA`, that is the only content allowed within the element. This doesn't mean that the location of an image is not important,

but only that this element is not meant to convey such information: the designers of this format assigned this responsability to the elements `INLINEMEDIAOBJECT` and `MEDIAOBJECT`, two ancestors of `IMAGEDATA` representing a media object within text streams or structured content flows, respectively.

This organization of the contents is widely used by schema designers, and embodies one of the principles of good design adopted in many different situations to increase the regularity, clarity and maintainability of the documents. These observations led me to consider the possibility of revising the model of patterns in order to distinguish between these two different uses of empty elements. In particular, I decided to split the Marker pattern into two more specialized ones: the Milestone pattern, that is meant to represent locations within the text content that are relevant for any reason, and the pattern Meta, that represents metadata elements that assert things about the document, but are not connected to its actual text content.

3.2.2 The Popup pattern

The patterns summarized in Section 3.1.2 are defined in terms of their *content models*, that indicate the structures or text nodes (possibly intermixed with each other) that an element can contain as well as their composition rules. The analysis presented in Section 3.2.1 revealed a strong connection between the function of empty elements (and, as a consequence, their characterization in terms of structural patterns) and their position within the document structure. This observation has led me to investigate the relevance of this aspect for other kinds of patterns. For this purpose, I first informally introduce the notion of *context* as the elements in which an element can appear.

There is a strong relation between the orthogonal dimensions of content models

and contexts. If an element A can contain an element B, in fact, two relations hold: B belongs to the content model of A, and A belongs to the context of B. This notion of context is therefore implicitly present in the definition of structural patterns, but what I want to do here is to investigate whether the explicit management of this aspect can improve the characterization of patterns, or emphasize structural features that have not been covered.

The first thing that has emerged from the analysis of this new perspective relates to the Table pattern. This pattern is defined as an ordered list used to group homogeneous objects into the same structure and, also, to represent repeating tabular data. According to this notion, the elements `footnote`, that is used to include a sequence of paragraphs to be relegated to a footnote or endnote, and `keywordset`, that serves as a container for homogeneous items (i.e. keywords) describing the paper, are both instances of the Table pattern, as confirmed by their DTD definition.

```
<!ELEMENT footnote      (para)+>
<!ELEMENT keywordset    (keyword+)>
```

The content model of these elements are very similar, and an analysis limited to this aspect would lead us to conclude that, from a structural point of view, they play the same role. By extending the analysis to their context, a remarkable difference clearly emerges: `footnote` can be contained within mixed content elements (i.e. `<emphasis>`, `<para>`, `<quote>`, `<subtitle>`, `<td>`, `<term>`, `<th>` and `<title>`), while the element `keywordset` is located into structures that do not contain text directly (i.e. `<info>`, a container for the metadata pertaining the document) and that gather a serie of elements under the same name or super-structure. The next example clarifies this distinction.

```
<!-- BalisageVol7-Sperberg-McQueen02.xml -->
```

```
<para>And the result has, of course, cost later working
   groups some indeterminate number of months or years
  <footnote><para>If we count person-months, the number is
     way too high, so we're talking about multiple person-
     years or person-decades of time.</para></footnote>
  trying to patch problems in the formal underpinnings of
  their specifications.</para>

<!-- BalisageVol7-Maloney01.xml -->
<para>The JATS schemas are downloadable from the NLM site,
   and come in flavors<footnote xml:id="foot-flavors">
    <para>"Flavor" is my term, which I haven't heard used
       anywhere else. I will use it throughout this paper to
       describe one of the main categories of JATS. One "
       flavor" roughly corresponds to one top-level DTD file,
        which might itself have several versions. In a
       detailed accounting, there are currently seven
       flavors:<variablelist><!-- OMITTED --></variablelist>
    </para>
    <para>This is somewhat complicated by the fact that the
       NISO standard versions of JATS use a different version
        numbering scheme, and so should also be considered
       separate flavors, even though they are really just
       newer versions of the existing NLM DTDs.</para>
  </footnote>, which have different semantics and use cases.
  These include Archiving<para>
```

```
<!-- BalisageVol10-Delpratt01.xml -->
<info>
  <confgroup><!-- OMITTED --></confgroup>
  <abstract><!-- OMITTED --></abstract>
  <author><!-- OMITTED --></author>
  <author><!-- OMITTED --></author>
  <legalnotice><!-- OMITTED --></legalnotice>
  <keywordset role="author">
    <keyword>XSLT</keyword>
    <keyword>Browser</keyword>
    <keyword>GWT</keyword>
    <keyword>Java</keyword>
    <keyword>JavaScript</keyword>
  </keywordset>
</info>
```

The element `footnote` is one of the most representative examples of a family of elements that are meant to include complex substructures that interrupt but do not break the main flow of text.

Authors make extensive use of these structures to inject comments, references and other more complex structured information within the text in close proximity to the content to which they refer. Two different mechanism are used to represent these common and widespread structures, the first suitable for static contexts such as printed documents, and the other for interactive contexts such as digital systems. In the first case, their content is extracted from the text, collected at the foot of the page or under a separate heading in specific sections, and replaced by superscripted symbols or markers that refers to the content. In dynamic contexts such as digital

libraries this approach is replaced by the pop-up technique: in this case, the element content is hidden until the users decide to activate it by clicking or hovering on the related mark in the text, and a small window with the content consequently appears ("pops up") in the foreground of the visual interface. Both of these solutions are quite popular, since they realize what these structures are meant for: not to interrupt the reader and, at the same time, to inform them about the presence of a related information the can be accessed or shown on their demand, respectively.

For all the aforementioned reasons, I decided to introduce a new pattern named Popup to the model of structural pattern presented in Chapter 4 to deal with similar cases.

3.2.3 The Field pattern

The notion of context is a fundamental component of the revised pattern theory: for instance, it provides a better characterization of relevant document structures such as empty elements and popups, as described in the previous sections. In this section I continue the analysis investigating the relation between the notion of context and other three patterns: Block, Inline and Atom.

The context is an important aspect to differentiate between instances of the Inline and Block patterns: indeed, these patterns share the same mixed content model, with the difference that blocks can't contain other block elements recursively (i.e. blocks are not listed in the allowed elements). This means that inline elements have mixed context (i.e. they can be contained in elements with mixed content models), while blocks can only be contained in elements that do not contain text. For the sake of brevity, from now on I will refer to this last type of context (i.e. elements that can contain markup elements but not text) as *bucket.*

The notion of context can also be used to differentiate between the two most common functions played by atom elements. This pattern is meant for both capturing the role of fragments within text streams, and for representing units of information that are assembled to build more complex structures. In the following excerpt[3], the elements `firstname` and `surname` are examples of the first scenario, and `biblioid` is an example of the second one:

```
<!ELEMENT firstname  (#PCDATA) >
<!ELEMENT surname    (#PCDATA) >
<!-- BalisageVol7-Sperberg-McQueen02.xml -->
<author>
  <personname>
    <firstname>C. M.</firstname>
    <surname>Sperberg-McQueen</surname>
  </personname>
  <personblurb><!-- OMITTED --></personblurb>
  <affiliation><!-- OMITTED --></affiliation>
</author>

<!ELEMENT biblioid    (#PCDATA) >
<!-- BalisageVol7-Sperberg-McQueen02.xml -->
<bibliography>
  <title>Bibliography</title>
  <bibliomixed xml:id="Coombs">Coombs, James H., Allen H.
     Renear and Steven J. DeRose. "Markup systems and the
     future of scholarly text processing." <emphasis
     role="ital">Communications of the ACM</emphasis> 1987
```

[3] BalisageVol7-Sperberg-McQueen02.xml

```
      Nov; 30(11):933-947. doi:<biblioid class="doi">
      10.1145/32206.32209</biblioid></bibliomixed>
  <!-- OMITTED -->
</bibliography>
```

Also in this case we can see that the different functions played by the elements correspond to different contexts: `firstname` and `surname`, in fact, appear in a bucket context, while `biblioid` is contained by a mixed content element. If we extend the analysis to their DTD definition, we can verify that this fact is not a coincidence, but it is a design choice made by the language developers, as shown in the following excerpt:

```
<!ELEMENT  personname  (firstname | surname | lineage |
   othername)*>
<!ELEMENT  bibliomixed  (#PCDATA | emphasis | link | quote |
   biblioid )*>
```

As we can see, the element `personname` (which is the only element that can contain `firstname` and `surname`) has a bucket content model, and the element `bibliomixed` (which is the only container allowed for `biblioid`) has a mixed content model. For this reasons, I decided to use two different patterns to handle these situations separately: the Atom pattern is limited to represent only boxes of text that are allowed in mixed elements, and the Field pattern (that I introduce here for the first time) is meant to represent units of text organized in more complex structures (i.e. they have a bucket context).

3.2.4 The Headed Container pattern

The structural patterns presented in section Section 3.1.2 includes three patterns that can be used to model all those structures where diversified objects need to be collected together: Container, Record and Table.

All of these patterns are quite similar, since they share the same content model (i.e. bucket), and they also have the same context (i.e. bucket)[4]. The Container pattern is the most general, and is defined as a set of heterogeneous, unordered, optional and repeatable elements of any kind, except inlines. The Record pattern is similar to Container, but it adds a constraint on the non-repeatability of the elements, and therefore can be considered as a specialization of the Container pattern. The Table pattern is another subclass of Container, since it adds the constraints that the elements must be homogeneous, and that they must be instances of the Atom, Block or Record patterns.

There is also another similar case that should be investigated: in fact, it is rather frequent to find containers whose content is preceded by one or more text wrappers for number, headers or bullets. A typical example is the element `section` from the Balisage Tag Set, whose content model is an heading composed of a title (mandatory) and zero or more subtitles, followed by the actual content of the element (i.e. block-level elements and sections).

```
<!ELEMENT  section  (
      title, subtitle?,          <!-- HEADING -->
      (%para.level;)*, section*  <!-- ACTUAL CONTENT -->
    )
```

[4]It's worth noting that all those situations where elements have bucket content and mixed context are instances neither of Container, nor Record, nor Table, since they are addressed by the Popup pattern by definition, as described in section Section 3.2.2

These structures are very important because they are often used as a major division or subdivision of the text, to define the skeleton of the document, and organize contents that are in some way related in groups of elements that concern the same topic or fulfill the same functional role within the document structure. Headed containers are often employed at any level of the hierarchy, and can nest the one into the other: the following fragment, for example, shows how the entire content of the document is organized around this pattern.

```
<!-- BalisageVol11-Bruggemann-Klein01.xml -->
<article>
  <title>Generating Schema-Aware XMLEditors in Xforms</title>
  <info><!-- METADATA --></info>
  <section>
    <title>Introduction</title>
    <para>In his PhD work...</para>
  </section>
  <section>
    <title>Architecture</title>
    <para>Components and their interactions are illustrated
       in...</para>
    <!-- THE CONTENT IS OMITTED -->
  </section>
  <!-- OTHER SECTIONS - OMITTED -->
  <section>
    <title>Related work</title>
    <para>We briefly discuss...</para>
  </section>
  <section>
```

```
    <title>Discussion, conclusions and further work</title>
    <para>Maalej in his PhD thesis...</para>
  </section>
  <section>
    <title>Acknowledgement</title>
    <para>The comments of the anonymous referees have been
       extraordinarily helpful. Thank you!</para>
  </section>
  <bibliography>
    <title>Bibliography</title>
    <bibliomixed xml:id="RecXForms1.1" xreflabel="Boy09">John
        M. Boyer, <emphasis>XForms 1.1</emphasis>, W3C
       Recommendation, W3C, October 2009
    </bibliomixed>
    <bibliomixed xml:id="XSDatatypes" xreflabel="BPM04">Paul
       V. Biron, Kaiser Permanente, and Ashok Malhotra,
       <emphasis>XML Schema Part 2: Datatypes Second Edition
       </emphasis>,W3C Recommendation, W3C, October 2004.
    </bibliomixed>
    <!-- OTHER BIBLIOGRAPHIC ITEMS -->
  </bibliography>
</article>
```

The element `article` at the top of the hierarchy is an headed container: in fact, it is composed by an heading with a title, followed by some metadata and a sequence of sections concluded by the bibliography. A step down in the hierarchy, `section` and `bibliography` also follow this pattern: the first gathers the main content of

the document, the latter the list of works referenced in the paper. As defined in the DTD, sections can be recursively nested too, creating subsections, sub-subsections, etc.

3.3 Towards a revised theory of structural patterns

In this section I discuss the results of the analysis described in Section 3.2, and examine how they relate to the two design principles that are the basis of the seven patterns described in Section 3.1.2: *syntactic minimality*, i.e. a few objects and composition rules are sufficient to express all the structures of a document, and *semantic expressiveness*, i.e. pattern-based documents make explicit the semantics of structures, relations and dependencies [36].

The first result of my investigation is the identification of four new patterns, some of which are brand new (Popup and HeadedContainer), while others arise from the refinement of other patterns (Milestone and Meta are the specialization of the Marker pattern, and the Field pattern derives from the Atom pattern). In the first analysis, one may think that this insertion increases the complexity of the model and, as a consequence, violates the principle of syntactic minimality. In order to clarify this point, it is important to discuss the differences between the objectives of the segmentation model and my work.

The pattern-based approach described in Section 3.1.2 is based on the idea that "any document can be segmented in some independent components and normalized into a pattern-based projection, that only uses a *very small set of objects and composition rules*" [36]. The minimality here is a strong requirement, because the

pattern-based model is the core of heterogeneous content management processes and, as a consequence, the complexity of these systems depends on the complexity of the model.

The objective of my work is to present a new approach to document analysis based on the idea of structural patterns. In my vision, structural patterns can be used to derive classes of elements persisting across documents and distilling the conceptualization of the documents and their components, and can give ground for automatic tools and services that do not rely on background information (such as schemas) at all. The basic idea is to leverage the structure of the document to extract information about the document itself. For example, this information can be used in support of the entity extraction and information synthesis tasks, to recognize the document components, to infer the logical organization of the document, to improve the effectiveness of NLP techniques on the content of the document, to study the rhetoric of the document and the structure of the argumentations, to derive citation networks, to investigate the document readability, etc.

For all these cases, the most important aspect of the pattern theory that I want to develop is not the minimality, but the ability to characterize in a *precise* and *complete* way the structure of the document. In practice, a trade-off between minimality/under-design and complexity/over-design must be considered:

- *under-design:* in the extreme case, we can imagine a theory consisting only of the Inline pattern. Given a document, we could interpret each element as an instance of the Inline pattern without violating such theory. Obviously, the semantic expressiveness of this model is null, because it does not provide any help either to design documents or to understand the document structure;

- *over-design:* on the other hand, the risk is to contemplate a set of pattern

too big. Given a document, we might consider, for example, the schema of the document as our pattern language. A similar theory of patterns would be useless, because it does not provide any further information to our analysis.

The pattern model that I intend to formalize should be placed in the middle of these two extremes, and be able to express, manage and represent in a clear and compact form the main information about the document structural semantics. In particular, the design of the revised pattern model is characterized by three properties:

- *coverage:* the ability to capture any possible situation in any document;
- *terseness:* the ability to represent the structure of a document with a small number of objects and composition rules;
- *expressiveness:* the ability to make explicit the semantics of structures, relations and dependencies.

In order to analyze the relation between structural patterns and these properties, it is important to first introduce some concepts that are the core of the pattern model. In particular, since in SGML and XML (and, in general, in all the meta-markup languages based on the OHCO model [34]) the semantics about the document structure derives from the containment relation, my work focuses on two dimensions: *content model*, which is derived from the direct application of the containment relationship, and the *context*, which is the inverse of the containment relation. By combining the possible content models[5] (i.e. *Marker*, *Flat*, *Bucket*,

[5]The possible content models are: *mixed* (textual content and other markup elements are allowed), *bucket* (only markup elements), *flat* (only textual content) and *marker* (neither markup elements nor textual content is allowed, i.e. they are empty elements).

Mixed) and contexts[6] (i.e. *Mixed* and *Bucket*) it is possible to obtain eight general patterns (i.e. *Milestone, Meta, Atom, Field, Popup, Containerm Inline, Blocks*) that are the basis of the pattern theory.

Table 3.3: The eight patterns of the revised pattern model. Any possible situation is covered by combining the four content models and the two contexts.

Context \ Content model	**MARKER**	**FLAT**	**BUCKET**	**MIXED**
MIXED	Milestone	Atom	Popup	Inline
BUCKET	Meta	Field	Container	Block

Coverage, terseness and expressiveness are closely related to the way in which patterns are defined. For instance, as shown in Table 3.3, any possible situation is covered by the resulting model[7]: in fact, there is exactly one pattern for each of the possibile situations within documents (coverage). Moreover, the complexity of any document is reduced to a few cases, i.e. the eight patterns (terseness). Finally, these patterns represent the major classes of structures used to organize documents, and each of them has a specific function and a precise characterization within the structure of the document (expressiveness). Another important aspect that testifies to the expressivity of patterns is the ability to develop useful application on top of the pattern-based representation of documents, as described in Chapter 6.

Altough the limited set of patterns is able to give a meaningful and explicit characterization of the document organization, other relations (e.g. order) and properties (e.g. cardinality) can be used to describe more specific behaviorss: for example,

[6]There are only two possible contexts: *mixed* (textual content and other markup elements are allowed) and *bucket* (only markup elements are allowed).

[7]In XML, it's very common to find element instances that occur in different *contexts*: in DocBook, for example, the element *citation* may occur in a paragraph surrounded/not surrounded by textual content – i.e. in a mixed/buckect context). Similarly, elements may have different *content*: in a DocBook document, it may happen that some instances of the *para* element contain both text and elements (i.e. they have mixed content), others only elements (i.e. they have bucket content), others only text (i.e. they have flat content). These situations are discussed in Section 5.1.1.

Container has a very general definition, and it is thus useful to define subclasses that describe situations all ascribable to the Container pattern, but that have a typicality worth of their own pattern. The class Container, in fact, can be specialized into at least three subclasses: HeadedContainer, Table, and Record.

Chapter 4

A revised theory of structural patterns

In this chapter I introduce a novel method to address document patterns, which generates, in my view, a systematic collection of interesting patterns by specifying a few meta-structures and some precise rules for combining them.

Patterns are organized around two orthogonal dimensions: their *content model* and their *context*. The *content model* indicates the structures or text nodes (possibly intermixed with each other) that an element can contain as well as their composition rules, while the *context* indicates the elements in which that element can appear. There is a strong relation between content models and contexts. If an element A can contain an element B, in fact, two relations hold: B belongs to the content model of A, and A belongs to the context of B. This constitutes the basis for the whole theory. Order relations of the elements of a content model should also be considered, and an example will be given in Section 4.2.

Instead of defining a large number of complex and diversified structures, we found a small number of *structural patterns* that are sufficient to express what most users need. The two main characterizing aspects of such set of patterns are:

- *orthogonality* – each pattern has a specific goal and fits a specific context. The

orthogonality between patterns makes it possible to assign a single pattern to each of the most common situations in document design. Conversely, for every situation a designer encounters in the creation of a new markup language, the corresponding pattern is immediately selectable and applicable;

- *assemblability* – each pattern can be used only in some contexts within other patterns. Far from being a limitation, this strictness provides expressiveness and non-ambiguity in the patterns. By limiting the possible choices, patterns prevent the creation of uncontrolled and misleading content structures.

Patterns allow authors to create unambiguous, manageable and well-structured documents. Also, thanks to the regularity they provide, it is possible to perform easily complex operations on pattern-based documents even when knowing very little about their vocabulary. Thus designers can implement more reliable and efficient tools, can make hypotheses regarding the meanings of document fragments, can identify singularities and can study global properties of sets of documents.

This chapter is organized as follows: in Section 4.1 I provide a formal definition of the eight patterns at the basis of the theory; in Section 4.2 I describe three situations that are ascribable to the Container pattern, but that have a typicality worth of their own, and describe three new patterns to model them. Since the pattern model is the result of a long project made by my research group, for the rest of this chapter I will use the first plural person to indicate the research group I belong to. The main contribution I gave to this part of the work is the bottom-up analysis on document instances presented in Chapter 3, which brought, for instance, to the identification of four new structural patterns, and the algorithm for the automatic pattern identification and its evaluation described in Chapter 5: both of these works have contributed to bring the pattern theory into the shape described in this chapter.

4.1 The Pattern Ontology: core model

This section provides a definition of the concepts at the basis of the theory of structural patterns. This layer consists of precise definitions of some properties of markup elements and their content. The whole theory is formally defined through description logic formulas [61] [71] and has been implemented as an OWL ontology [79] available at http://www.essepuntato.it/2008/12/pattern. The choice of description logic (DL) was mainly due to the application environment in which such meta-structures are further processed. As I discuss later, in fact, I have developed an engine that recognizes these patterns by exploiting Semantic Web technologies and OWL-DL reasoning capabilities, which work on axioms of description logic. The transparent integration with Semantic Web data was another key factor for using OWL-DL, which allows the combination of the identification of meta-structures, as performed through our ontology, with other sources of information so as to validate content at different levels of abstraction and to perform sophisticated queries and analyses, such as studying peculiarities of the documents and their editing processes.

4.1.1 Basic properties of content models and contexts

Markup elements are first organized in abstract classes from which the actual patterns are derived. At the abstract level, markup elements can be organized in four disjoint classes according to their ability to contain text and/or other elements.

We define *Textual* the class of markup elements that *can have textual content* in their content models and *NonTextual* (clearly disjoint with Textual) the class of elements that cannot. We also define *Structured* the class of markup elements that can *contain other markup elements*, and *NonStructured* as the class of elements that

cannot. These two classes are disjoint[1].

$\texttt{Textual} \sqsubseteq \top$

$\texttt{NonTextual} \sqsubseteq \top$

$\texttt{NonTextual} \equiv \neg\ \texttt{Textual}$

$\texttt{Textual} \sqcap \texttt{NonTextual} \sqsubseteq \bot$

$\texttt{Structured} \sqsubseteq \top$

$\texttt{NonStructured} \sqsubseteq \top$

$\texttt{NonStructured} \equiv \neg\ \texttt{Structured}$

$\texttt{Structured} \sqcap \texttt{NonStructured} \sqsubseteq \bot$

We define the property *contains* (and its inverse *isContainedBy*) on *Structured* to indicate the markup elements its individuals contain:

$\exists\texttt{contains}.\top \sqsubseteq \texttt{Structured}$

$\texttt{isContainedBy} \equiv \texttt{contains}^-$

By combining the four classes defined above we are able to generate four new classes:

- class *Marker*. Individual of this class can contain neither text nodes nor elements.

- class *Flat*. Individual of this class can contain text nodes but no elements;

[1]The pattern theory is introduced by means of description logic (DL) formulas. I briefly introduce the DL notation in order to help readers in reading the formalities of our theory: "$\top$" and "$\bot$" refer to the *top concept* (i.e. the concept with every individual as instance) and *bottom concepts* (i.e. the empty concept); "$\sqsubseteq$" expresses *concept inclusion*; "$\equiv$" expresses *concept equivalence*; "$\neg$", "$\sqcup$" and "$\sqcap$" express *negation*, *disjunction* (i.e. union) and *complement* respectively; "-" expresses the *inverse role*, while "$\exists$" and "$\forall$" express *existential* and *universal restrictions*; "$\leq$" express the *at-most restriction* ("$\leq$nR" refers to the set of individuals that are related, through a relation R, to at most n of other individuals); ":" expresses a *value restriction*, ("R:v" is the set of individuals that are related, through a particular relation R, to a specific value*)*; for more details, see [61] and [71].

- class *Bucket*. Individual of this class can contain other elements but no text nodes;
- class *Mixed*. Individuals of this class can contain other elements as well as text nodes;

These classes are defined as follows and shown together with their superclasses in Fig. 4.1.

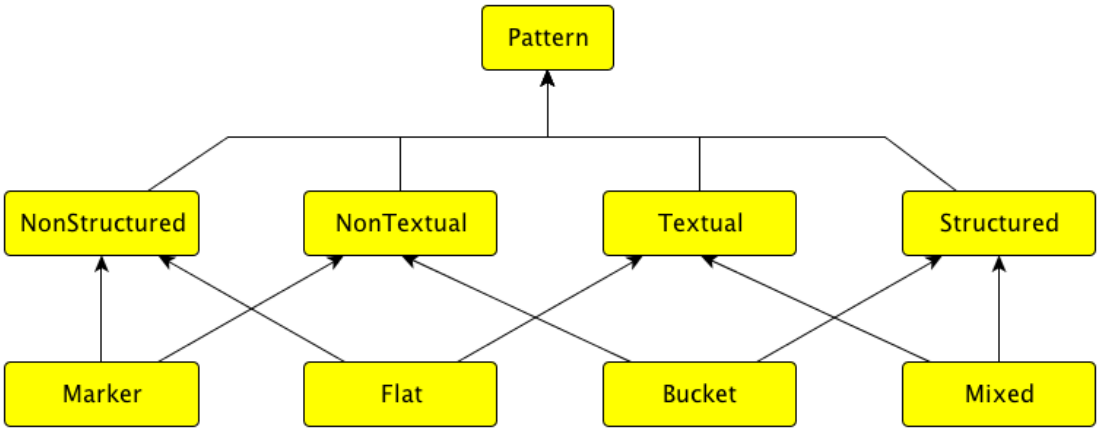


Figure 4.1: The abstract classes defining the hierarchical structure structural patterns are derived from. The arrows indicate sub-class relationships between patterns (e.g. Mixed is sub-class of Structured).

$$\texttt{Marker} \sqsubseteq \texttt{NonTextual} \sqcap \texttt{NonStructured}$$

$$\texttt{Flat} \sqsubseteq \texttt{Textual} \sqcap \texttt{NonStructured}$$

$$\texttt{Bucket} \sqsubseteq \texttt{Structured} \sqcap \texttt{NonTextual}$$

$$\texttt{Mixed} \sqsubseteq \texttt{Structured} \sqcap \texttt{Textual}$$

The behaviour of the content models can be fully described by these classes. Contexts can be characterized in a similar way, with the only important difference that, since each element clearly appears only in a content model that accepts elements, i.e., in a structured content model, the context of an element can only be

either *Mixed* or *Bucket,* depending on whether it contains text or not.

4.1.2 Structural patterns

The combination of the possible content models (i.e. *Marker*, *Flat*, *Bucket*, *Mixed*) and contexts (i.e. *Mixed* and *Bucket*) bring the number to the identification of distinct patterns to eight. The abstract classes of the ontology, in fact, can be specialized into eight concrete patterns as shown in Fig. 4.2. Notice that, for each pair of patterns derived from the same abstract class, the left one has a *Mixed* context and the right one a *Bucket*.

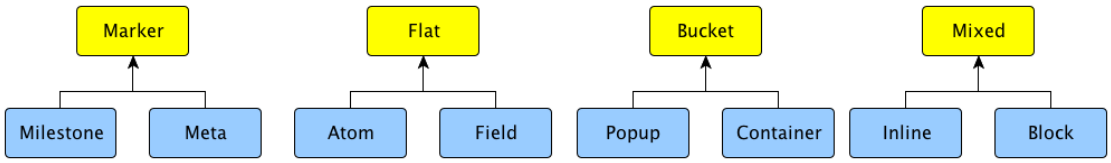


Figure 4.2: The eight concrete patterns derived from the abstract classes of the ontology. The arrows indicate sub-class relationships between patterns.

The relations between patterns, their content models and their contexts has been summarized in the table in Section 3.3, that are repeated here in Fig. 4.3 for convenience.

Content model / Context	MARKER	FLAT	BUCKET	MIXED
MIXED	Milestone	Atom	Popup	Inline
BUCKET	Meta	Field	Container	Block

Figure 4.3: The eight patterns classified according to the particular content model and context they have.

Table 4.1 instead summarizes all patterns giving a brief description of their goal and some examples from HTML and DocBook vocabulary [121].

Table 4.1: The eight structural patterns for descriptive documents.

Pattern	Description	HTML	DocBook
Milestone	Any content-less structure (but data could be specified in attributes) that is allowed in a mixed content structure but not in a container. The pattern is meant to represent locations within the text content that are relevant for any reason.	br	xref, co
Meta	Any content-less structure (but data could be specified in attributes) that is allowed in a container but not in a mixed content structure. The pattern is meant to represent metadata elements that assert things about the document, but are disconnected from its actual text content.	meta, cols, colspan, area	imagedata, colspec
Atom	Any simple box of text, without internal substructures (simple content) that is allowed in a mixed content structure but not in a container.	-	email, code
Field	Any simple box of text, without internal substructures (simple content) that is allowed in a container but not in a mixed content structure.	title	pubdate, publishername
Popup	Any structure that, while still not allowing text content inside itself, is nonetheless found in a mixed content context. The pattern is meant to represent complex substructures that interrupt but do not break the main flow of the text, such as footnotes.	-	footnote, tip
Container	Any container of a sequence of other substructures and that does not directly contain text. The pattern is meant to represent higher document structures that give shape and organization to a text document, but do not directly include the content of the document.	html, body, table, map	bibliography, preface
Inline	Any container of text and other substructures, including (even recursively) other inline elements. The pattern is meant to represent inline-level styles such as bold, italic, etc.	b, i, a, span	emphasis
Block	Any container of text and other substructures except for (even recursively) other block elements. The pattern is meant to represent block-level elements such as paragraphs.	p, div, address	para, caption

We give now a formal characterization of these patterns and their relations. In the following subsection we also describe some specializations of the *Container*

pattern that occur frequently.

The first two patterns are used for the elements that contain neither other elements nor textual content. We in fact define two subclasses of the class *Marker: Milestone* and*Meta.*

The elements of the *Milestone* pattern are empty and can only be contained within mixed elements. The formal definition of this class is as follows:

$$\texttt{Milestone} \equiv \texttt{Marker} \sqcap \forall \texttt{isContainedBy.Mixed}$$

$$\texttt{Milestone} \sqsubseteq \exists \texttt{isContainedBy.Mixed}$$

Since *Milestone* elements are surrounded by text nodes, their distinctive characteristic is the *location* they assume within the document. Examples of DocBook elements typically used as compliant with the *Milestone* pattern are *xref* and *co*.

The class *Meta* characterizes empty elements that are placed in a content-only context. Unlikely *Milestones*, their main characteristic is their mere *existence*, independently from the position they have within the document. *Meta* elements often convey information about the whole document or specific parts of it, independently of their position (e.g., the elements *imagedata* or *colspec* in DocBook). *Meta* elements can be contained only within *Bucket* elements, formalized as follows:

$$\texttt{Meta} \equiv \texttt{Marker} \sqcap \forall \texttt{isContainedBy.Bucket}$$

$$\texttt{Meta} \sqcap \texttt{Milestone} \sqsubseteq \bot$$

Other patterns are used for the elements that can contain text but no other elements. They specialize the class *Flat* in our ontology.

Atom is the class of elements that contain only literal text (and no other elements) within the document body. Similarly to *Milestone*, elements of the *Atom* pattern can only be contained within mixed elements (and consequently they also cannot be used as root elements of documents).

$$\texttt{Atom} \equiv \texttt{Flat} \sqcap \forall \texttt{isContainedBy}.\texttt{Mixed}$$

$$\texttt{Atom} \sqsubseteq \exists \texttt{isContainedBy}.\texttt{Mixed}$$

The class *Field* describes literal metadata or text that is not really part of the document body, differently from its disjoint sibling *Atom*. *Field* is similar to *Meta* but the main difference is that *Field* can contain textual content, while *Meta* cannot:

$$\texttt{Field} \equiv \texttt{Flat} \sqcap \forall \texttt{isContainedBy}.\texttt{Bucket}$$

$$\texttt{Field} \sqcap \texttt{Atom} \sqsubseteq \bot$$

Examples of DocBook elements typically used as compliant with the *Field* pattern are *pubdate* and *publishername*.

The class *Bucket* is specialized into two subclasses to be used for complex structures: *Popup* and *Container*. *Popup* is the class of elements that is only present within mixed elements (and consequently they also cannot be used as root elements of documents) but only contain other elements. Elements following this pattern have no textual content and contain only elements compliant with the patterns *Meta*, *Field*, *Block* (that will be introduced in the following) and *Container*, as shown in the following excerpt:

$$\texttt{Popup} \equiv \texttt{Bucket} \sqcap \forall \texttt{isContainedBy}.\texttt{Mixed}$$

$$\texttt{Popup} \sqsubseteq$$

$$\forall \texttt{contains}.(\texttt{Container} \sqcup \texttt{Field} \sqcup \texttt{Meta} \sqcup \texttt{Block}) \sqcap$$

$$\exists \texttt{isContainedBy}.\texttt{Mixed}$$

Popup elements are used whenever complex structures need to be placed within content elements such as paragraphs. Examples of DocBook elements typically used in a way compliant with the *Popup* pattern are *footnote* and *tip*.

The sibling pattern *Container* concerns the structural organization of a document. Elements following this pattern contain no textual content and contain only

elements compliant with the patterns *Meta*, *Field*, *Block* and *Container*. *Container* shares the same content model of *Popup* but they may be contained only in bucket elements, which makes these classes disjoint. Its formalisation is as follows:

$$\texttt{Container} \equiv \texttt{Bucket} \sqcap \forall \texttt{isContainedBy.Bucket}$$

$$\texttt{Container} \sqsubseteq \forall \texttt{contains.(Container} \sqcup \texttt{Field} \sqcup \texttt{Meta} \sqcup \texttt{Block)}$$

$$\texttt{Container} \sqcap \texttt{Popup} \sqsubseteq \bot$$

Examples of DocBook elements typically used as compliant with the *Container* pattern are *bibliography* and *preface*.

The last two classes are derived from the abstract class *Mixed* and are meant to be used where text nodes are mixed with elements that are further nestable: *Block* and *Inline*.

Block is the class that organises the document content as a sequence of other nestable elements and text nodes. Elements of the class *Block* can contain text and other elements of patterns *Inline*, *Atom*, *Milestones* and *Popup* it is a requirement that they are contained only within *Bucket* elements:

$$\texttt{Block} \equiv \texttt{Mixed} \sqcap \forall \texttt{isContainedBy.Bucket}$$

$$\texttt{Block} \sqsubseteq \forall \texttt{contains.(Inline} \sqcup \texttt{Atom} \sqcup \texttt{Milestone} \sqcup \texttt{Popup)}$$

Inline is the class of elements that have the same use and content model of the pattern *Block*, but differing primarily because:

- they can contain other elements of the same pattern (block elements cannot);
- they can only be contained in *mixed* elements, i.e., inline and blocks.

These constraints imply that inline elements cannot be used as root elements of documents and that *Block* is disjoint with *Inline* (i.e., a markup element cannot be a block and an inline at the same time):

$$\texttt{Inline} \equiv \texttt{Mixed} \sqcap \forall \texttt{isContainedBy}.\texttt{Mixed}$$

$$\texttt{Inline} \sqsubseteq \forall \texttt{contains}.(\texttt{Inline} \sqcup \texttt{Atom} \sqcup \texttt{Milestone} \sqcup \texttt{Popup}) \sqcap \exists \texttt{isContainedBy}.\texttt{Mixed}$$

$$\texttt{Block} \sqcap \texttt{Inline} \sqsubseteq \bot$$

4.2 The Pattern Ontology: specializations of the Container pattern

Container has a very general definition. It is thus useful to define subclasses that describe situations all ascribable to the Container pattern, but that are distinctive enough to merit their own distinct pattern. The class *Container*, in fact, can be specialized into at least three sub-classes (*HeadedContainer*, *Table* and *Record*) as shown in Fig. 4.4.

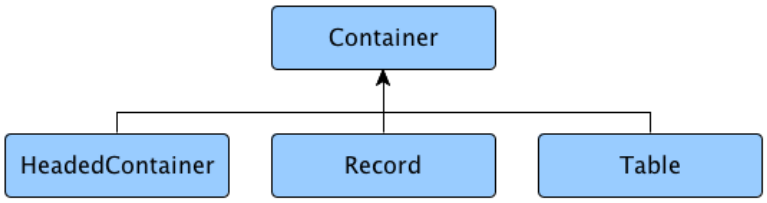


Figure 4.4: The three subclasses of the class Container.

Table 4.2 on the next page describes briefly these patterns and reports some examples in HTML and DocBook. Their formalisation follows in this section.

While the content model of structured elements (i.e. mixed and bucket elements) can contain any kind of optional and repeatable selection of elements, we need to be able to define some restrictions to their element repeatability so as to characterise

Table 4.2: The three sub-patterns of the Container pattern.

Pattern	Description	HTML	DocBook
Record	Any container that does not allow substructures to repeat themselves internally. The pattern is meant to represent database records with their variety of (non-repeatable) fields.	html	address, revision
Table	Any container that allows a repetition of homogeneous substructures. The pattern is meant to represent a table of a database with its content of multiple similarly structured records.	ul	keywordset
Headed Container	Any container starting with a head of one or more block elements. The pattern is usually meant to represent nested hierarchical elements (such as sections, subsections, etc., as well as their headings). This is the only pattern we use that requires the specification of an order in the sequence of the components.	-	section, chapter

the specialisation of the *Container* pattern. We thus define the boolean properties *canContainHomonymousElements,* true if the element can contain elements that share the same name[2] of XML elements., and *canContainHeteronymousElements*, true if an element can contain elements with different names. In addition, we define *containsAsHeader* as a sub-property of *contains* to specify when a structured-based element contains *header* elements.

$$\exists \texttt{canContainHomonymousElements}.\top \sqsubseteq \texttt{Structured}$$

$$\top \sqsubseteq \leq 1\,\texttt{canContainHomonymousElements}$$

$$\exists \texttt{canContainHeteronymousElements}.\top \sqsubseteq \texttt{Structured}$$

$$\top \sqsubseteq \leq 1\,\texttt{canContainHeteronymousElements}$$

$$\texttt{containsAsHeader} \sqsubseteq \texttt{contains}$$

Through these new properties, we can define, among many, three subtypes of the *Container* pattern that we found particularly useful. For instance, the pattern *Record* captures the characteristics (typical of database records) of having many

[2]By *name* we mean the pair (*namespace*, *generic identifier*)

differently named elements with no repetitions. As such, its element can only contain heteronymous and non-repeatable elements, as in the following axioms:

```
Record ≡
  Container ⊓
  canContainHomonymousElements:false ⊓
  canContainHeteronymousElements:true
```

Examples of DocBook elements typically used as compliant with the *Record* pattern are *address* and *revision*.

On the opposite end, we find elements that allow a repetition of elements of the same name, as a database table allowing many homogeneous records. For this reason we call this pattern *Table*. Elements compliant with the *Table* pattern must contain only homonymous elements (that can be repeated), as follows:

```
Table ≡
  Container ⊓
  canContainHomonymousElements:true ⊓
  canContainHeteronymousElements:false
```

Representative DocBook elements that are commonly used as compliant with the pattern *Table* are *keywordset*.

Finally also rather frequent in documents is the pattern where content is preceded by one or more text containers for numbers, headers or bullets. It is interesting to note that, in our experience, this is the most general case in which the order of the elements of a content model is relevant. We call *HeadedContainer* the subclass of *Container* whose content model begins with one or more block elements (the heading), as specified through the property *containsAsHeader* :

```
HeadedContainer ⊑ Container ⊓ ∀containsAsHeader.Block
```

Examples of DocBook elements typically used as compliant with the *Headed-Container* pattern are *section* and *chapter*.

Finally, it is also important to require that these subclasses of *Container* are all reciprocally disjoint, as follows:

$$\texttt{Table} \sqcap \texttt{Record} \sqsubseteq \bot$$

$$\texttt{HeadedContainer} \sqcap \texttt{Record} \sqsubseteq \bot$$

$$\texttt{HeadedContainer} \sqcap \texttt{Table} \sqsubseteq \bot$$

Of course this is by far not a complete selection of the possible or the useful subclasses of containers, but are found quite frequently in real documents and for this reason they were identified and named. All other variations in the use of the Container pattern will be categorized simply as Containers.

Chapter 5

Recognising structural patterns in XML-based documents

In order to verify whether the theory of patterns presented in the previous chapter is adequate and complete, I describe here an algorithm for the automatic identification of structural patterns in XML documents that relies on no background information about the vocabulary, its intended meaning and its schema. This algorithm takes as input a set of XML documents using the same vocabulary, and produces as output a pattern scheme, that is a list of associations element-pattern.

It's worth noting that, in the last step of the computations, the algorithm exploits Semantic Web technologies to verify that the results comply with/validate the results of the analysis against the theory of patterns. The basic idea is to process with an OWL reasoner the Pattern Ontology (TBOX) and the EARMARK representation of the documents enriched with the information about the pattern assignments (ABOX) in order to check the overall consistency. This method is similar to the approach described in [42] to validate markup documents against syntactical constrainst expressed in schema.

Finally, I present an experiment I performed on eight different vocabularies for

a total of more than 1100 documents. The main objectives of this test are to check the adequacy and completeness of the theory of patterns, and to verify whether the characterization provided by a pattern-based analysis can provide valuable insights for comparing different languages.

This chapter is organized as follows. In Section 5.1 I introduce the concepts of coherency, pattern shift, partition and pattern scheme that are the basis of the algorithm for the automatic identification of structural patterns. In Section 5.2 I describe a language-independent algorithm that assigns patterns to the elements of XML documents. In Section 5.3 I evaluate the algorithm on a set of 1100 documents from eight different vocabularies.

5.1 Assigning patterns to documents

The theory introduced in the previous chapter allow us to assign one specific pattern to each element of a document, by analysing its local content model and context. Let us introduce an example to clarify this issue[1]:

```
<section>
  <title>Available physical types</title>
  <para>As a result of a query execution ...</para>
  <para><emphasis role="ital">Note:</emphasis> The preceding
     subsection introduced the notion of physical types...
  </para>
  <table>
    <caption>
```

[1]The full version of the document is available online at the address http://www.balisage.net/Proceedings/vol5/xml/Rennau01/BalisageVol5-Rennau01.xml. Some content has been removed for the sake of clarity.

```
      <para>
        <emphasis role="bold"><emphasis role="ital">Summary
           of Java types delivered by XQJ.</emphasis>
        </emphasis>
      </para>
    </caption>
    <col align="left" valign="top" span="1"/>
    <thead>
      <tr valign="top">
        <th align="left" valign="top">category</th>
        <th align="left" valign="top">types</th>
      </tr>
    </thead>
    <tbody>
      <tr valign="top">
        <td><emphasis role="bold">atomic types</emphasis>
        </td>
        <td>Boolean, BigDecimal, BigInteger, ...</td>
      </tr>
      <tr valign="top">
        <td><emphasis role="bold">node types</emphasis></td>
        <td>Document, Element, Attr, <!-- OMITTED --> </td>
      </tr>
    </tbody>
  </table>
  <para> <!-- OMITTED --> </para>
</section>
```

I can identify the content model (CM) and context (CTX) of all the markup elements in the previous excerpt, and consequently their actual structural patterns:

- the element *section* is a HeadedContainer (CM = Bucket with element *title* as heading, CTX = Bucket, since it is contained in the document element *article*);
- the element *title* is Block (CM = Mixed, CTX = Bucket);
- the first *para* child of *section* is Field (CM = Flat, CTX = Bucket), thesecond *para* child of *section* is Block (CM = Mixed, CTX = Bucket) and the last *para* child of *caption* is Container (CM = Bucket, CTX = Bucket);
- the first *emphasis* is Atom (CM = Flat, CTX = Mixed), the second is Container (CM = Bucket, CTX = Bucket), and the last three are Field (CM = Flat, CTX = Bucket);
- the element *table* is a Record (CM = Bucket of heteronymous elements, CTX = Bucket);
- the elements *caption* and *thead* are Container (CM = Bucket, CTX = Bucket)
- the elements *col* are Meta (CM = Marker, CTX = Bucket).
- the elements *tbody* and *tr* are Tables (CM = Bucket of Homonymous elements, CTX = Bucket);
- the first and the third *td* elements are Container (CM = Bucket, CTX = Bucket);
- the second and forth elements *td* are Field (CM = Flat, CTX = Bucket).

This example shows that, athough in most cases the pattern assignment is clear, there are some elements (i.e. *para*, *emphasis* and *td*) that can be associated with more than one pattern. In the following sections I investigate these ambiguous situations.

5.1.1 Coherency and pattern shifts

As seen in the previous section, individual assignments may generate inconsistencies, where the same element in different parts of the document is assigned to different patterns. These inconsistencies are often legitimate and solvable, although in other cases they are more complex to deal with.

Definition 1: local coherency. An element E is *locally coherent* if all its instances in a document share the same structural patterns, otherwise it is *locally incoherent*. For instance, in the previous excerpt the elements *caption*, *col*, *section*, *table*, *thead*, *tbody* and *tr* were locally coherent, while the elements *para*, *td* and *emphasis* were locally incoherent.

Of course, the previous definition can be also applied to a set of documents rather than just one, so we need a broader definition:

Definition 2: global coherency. An element E is *globally coherent* according to a set of documents S if all its instances in the set S have the same structural patterns. Of course, the global coherency of an element implies its local coherency within any document in the set.

The local or global incoherency is not a problem *per se*. In some cases it is possible to consider a different pattern for an element, so that its incoherency is reduced or completely eliminated. I call these pattern modifications ***shifts***.

Consider, for instance, two HTML documents both containing the element *strong*.

In one case all occurrences of *strong* contain plain text, so that *strong* is classified as Atom. In the other document, some instances of strong contain both text and an *emph* element, thereby they are classified as Inlines. The presence or absence of further elements within strong does not imply that the element is meant to cover two different needs in the two documents. It just depends on the specific content of each of them. Thus, we can shift the first assignment from Atom to Inline, achieving a global correct coherency. Notice that the same shift could also be valid within one single document.

Other shifts are also possible. For instance, an element that is recognized as Field in some documents and as *meta* in others can be shifted into a Field. That means that some information is missing in the second case, but all occurrences can be considered as empty fields without loss of information. Similar considerations can be extended to Fields and Blocks. Consider for instance the case of a *title*. In most cases *title* contain plain text (and the corresponding element is probably classified as Field), while in others they also include in-line elements such as bold or italic formatting (and are classified as Blocks). Shifting into Block in both cases is legitimate and increases global coherency.

It is also possible that more than two patterns are assigned to the same markup item within the same document. Consider for example the element *td* that represents a cell of an HTML table: it is possible that the majority of the cells within a table contain only plain text and therefore are recognized as Field, whereby other cells contain only elements (such as images, links, etc.), and as such are classified as Containers yet other cells are completely empty so that they are assigned to pattern Meta. This situation is handled by shifting all the elements to the most general case, i.e., in this case, the pattern Block.

All the admissible shifts are indicated with arrows in Fig. 5.1. They allow us to change the content model of an element without changing its context. Shifts that go in the opposite direction are not valid, as they would lose information. For instance, they would not consider the presence of text when shifting back from Bucket to Marker.

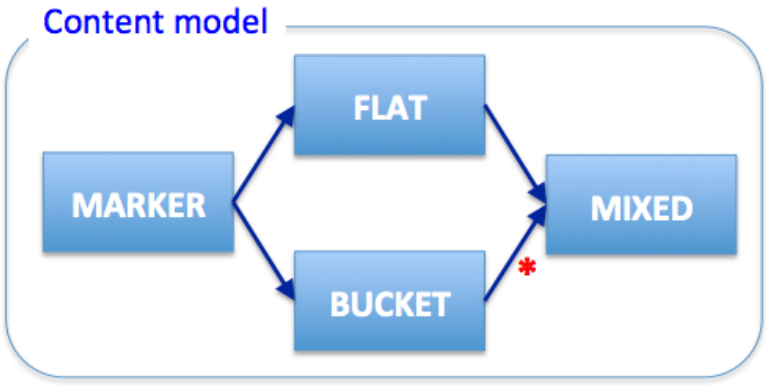


Figure 5.1: All the acceptable shifts. The asterisk as label of the arrow between Bucket and Mixed refers to a particular case of shifts, called *shifty-shifts*, which are still possible even if they change drastically the context (and, thus, the pattern) of all the elements contained by the shifted one.

I define the coherency obtained through shifts as follows.

Definition 3: coherency by shifting. A (globally or locally) incoherent element E is (globally or locally)*coherent by shifting* if all its instances have the same context, and their content models can be *acceptably* shifted so as to reach the same content model.

There is a particular kind of shifts called *shifty-shifts*, labelled with an asterisk in Figure 5, which is particularly delicate to address. A shifty-shift from Bucket to Mixed applied on an element E actually changes the context of all the children of E, and may change radically the structure of a document. It is the case of the elements *td* in the previous example, which can be shifted to Blocks and, thus, modify the nature of all the elements *para* they contain from Block to Inline. Were

this to happen to an element in the higher levels of the document hierarchy, even within a single document of a large set, it would completely disrupt the nature of all documents, whereby, for instance, the document element becomes the only Block and everything inside it becomes an indistinguishable Inline.

5.1.2 Pattern schemes and partitions

Once I have identified the patterns of the several element instances of a set of XML documents, I can group all the mappings *element instance-pattern* according to *pattern schemes*:

Definition 4: pattern scheme. Given a finite set of XML documents D, a *pattern scheme* S_D is the set of all the mappings from element instances in D to patterns.

Of course, pattern schemes can contain locally/globally coherent/incoherent mappings according to the situations encountered. In these cases, e.g., in the presence of global incoherency (but overall local coherency), I can generate two or more pattern sub-schemes by partitioning the set of documents so as to reach global coherency in each subset.

Definition 5: partition. A *partition* of a pattern scheme S_D is a set of pattern schemes S_{Di} where each D_i belongs to the same partition of D and each S_{Di} is globally coherent.

Of course, the presence of locally incoherent documents prevents partitions to even exist (there would be at least one globally incoherent sub-scheme), but, banning this situation, I can verify whether there are partitions of the scheme that are actually adopted by large set of authors of a document set.

5.2 An algorithm for the automatic recognition of structural patterns

In this section I describe an algorithm[2] that assigns patterns to the elements of one or many XML documents (using the same vocabulary) relying on no background information about the vocabulary, its intended meaning and its schema. The overall process assigns first a structural pattern to each element in the document trying to achieve *local coherency* or, if necessary, *coherency-by-shifting*, and then tries to achieve *global coherency*, possibly by applying even more shifts. If this is not possible it stops prompting the user to identify possible partitions of the dataset. The goal is to understand to what extent patterns are used in that set of documents.

The first part of the algorithm takes as input a single XML document. If the algorithm manages to obtain local coherency it succeeds, otherwise it returns pointers to the elements that generate inconsistencies. The overall process is performed in five steps:

Identification of potential content models and contexts. In this step I identify which of the four possible content models – empty (i.e. Marker), only text (i.e. Flat), only element (i.e. Bucket), both text and element (i.e. Mixed) – can be associated with each element, and thereby to identify the context for each of its children. This is the place where shifts come into play: whenever different occurrences of the same element appear to have different content models it tries to generalize them in a single one.

Pattern assignment. Next, a pattern is assigned to each element instance, starting from the content model and context identified in the previous step. This is

[2]The source code of the algorithm is available online at `http://fpoggi.web.cs.unibo.it/patterns/`

a direct application of the rules summarized in Figure 3.

Local coherency check. Next, a check is performed to verify whether the document has reached local coherency after the pattern assignments. To do so, it is sufficient to verify that all instances of the same element have been assigned to the same pattern. Notice that two instances of the same element will always have the same potential content model (since it has been derived by shifting on all instances) but can still have different contexts when used in different locations. If no further shifts are possible, the algorithm concludes that the document is locally inconsistent and reports the elements generating the problem.

Container specialization. This step is meant to identify the three subclasses of Container (Table, Record, and HeadedContainer) by following the rules discussed in Section "Specialisations of the Container pattern". The algorithm uses the data collected so far in order to discern the Container elements: it retrieves all instances of *Container*, groups them by the name of the element and checks which specialization rules can be applied. If none of these rules can be applied, the element remains a Container. There is a borderline case worth discussing, in which every element of a group has only one child node and these children nodes have the same name. These elements therefore lie at the intersection of the pattern *Table* and *Record,* and are arbitrarily assigned to the pattern *Table*.

The opposite operation, ***container generalization***, can be performed as a step towards global coherency: generalizing *Record*s, *Table*s, *HeadedContainer*s into simple *Container*s. Consider, for instance, the case of an element recognized as *record* in some documents and *table* in others. That might happen because the element is meant to collect heterogeneous information but, in some cases, it contains several different elements with no repetitions while in others it only contains only

one element (and is therefore recognized as *Table*). It is therefore appropriate to generalize these patterns as *Containers*. Of course generalizing to containers is in a way to surrender the specialization of the containers and accept that some containers simply cannot be generalized. Fortunately, the recourse to this operation has been restricted to just a few well-justified situations.

Validation. The last step consists in verifying whether the associations between elements and patterns are valid. This is actually an optional step, just added to improve reliability and to double-check the final output. As described in [42], this test can be performed easily using the technologies of the Semantic Web in three steps:

- converting the XML document given in input in EARMARK [42] a version of the conversion tool is available online[3]);
- associating the previously calculated pattern to each element (through a *rdf:type* assertion);
- launching a reasoner such as Pellet [102] or Hermit [96] to check if the Pattern Ontology[4] with these added assertions (the EARMARK document and the pattern associations) is consistent (all the pattern constraints hold) or not (there are some errors when assigning patterns to elements).

Once the algorithm just described is applied to each document in a dataset, documents locally incoherent are discarded and global coherency of the remaining ones is verified by comparing each execution against each other and by applying, where possible, the aforementioned global shifts, including container generalization as explained.

[3]http://www.essepuntato.it/xml2earmark
[4]Pattern Ontology: http://www.essepuntato.it/2008/12/pattern.

5.3 Evaluation: checking patterns on live documents

The operation of automatic recognition of the structural pattern described in the previous section is independent from the markup language of the documents taken into account and, consequently, from their schema. I mean to focus on how most *authors* of documents actually choose their markup, rather than on how the *designers* of the schema give room to special needs of a small number of authors.

For instance, the development of vocabularies used by large communities such as TEI [113] and DocBook [121] has been (and still is) a long process that had to deal with complex constraints: schema designers are often required to capture all requirements of their prospective users, covering very heterogeneous situations and foreseeing any potential validation mismatch or misinterpretations. These difficulties conspire to produce rich, complex schemas, that require time and effort to be fully understood and applied, but that have an extremely simple core.

I rather propose to analyze the characteristics that emerge from real markup documents, not preventing any peculiar use of the elements still allowed by the schema, but trying to go for the simple core of the language as it is actually used by the majority of document authors. In particular I want to check if the theory based on eight simple structural patterns is able to capture and summarize the guidelines used by the authors of markup documents in their independent everyday practice.

How do real documents perform compared against the theory of patterns? In this section I discuss the e, together with instructions on how to run thexperimental results of tests runs of my algorithm to determine the actual use of patterns by document authors. These tests largely confirm my hypotheses, but raise some

unexpected issues.

In order to build a representative data set I collected about 1200 documents from eight different XML vocabularies. Vocabularies cover very different domains: from literary documents to technical documentation, from web pages to databases dumps, from conference proceedings to cookbooks. Documents vary a lot in terms of size and number of elements, and they were downloaded from very heterogeneous sources, all freely accessible on the Web. Table 5.1 on the next page briefly describes the sets of documents I studied, while full sources and the outcomes of my experiments are available at `http://fpoggi.web.cs.unibo.it/patterns/`.

I evaluated each group of documents separately. My goal was to study to what extent the structural organization of those documents was close to my pattern-based meta-model. To do so, I ran a Java implementation of the algorithm presented in the previous section on each paper, and then compared and combined these results for the overall dataset.

Results are encouraging. A summary view is given in Fig. 5.2 on page 93, where each point corresponds to a set of documents.

The Y-axis indicates the percentage of documents recognized as locally coherent (whose elements are all locally coherent). In half of the sets the assignment of patterns was complete and straightforward for all documents. In others I found several locally incoherent documents but most of the elements of those documents were still used according to my patterns. The X-axis, in fact, indicates the percentage of elements of the vocabulary that are globally coherent (i.e. are associated with one single pattern of my model). In six sets over eight the authors used more than 80% of the elements in a pattern-based fashion.

Moreover, a detailed analysis of discrepancies shows that most of the mismatches

Table 5.1: The full dataset used to evaluate patterns.

	Vocabulary	Source	#files	Min size (bytes)	Max size (bytes)	Avg size (bytes)	#elements
1	GXL	GXLThe full set of examples in the official documentation of GXL 1.0, available at: http://www.gupro.de/GXL/	21	578	32635	4733	14
2	RecipeML	RecipeMLThe first five archives of recipes from the Squirrel's RecipeML Archive, available at: http://dsquirrel.tripod.com/recipeml/↩ indexrecipes2.html	498	895	7196	2526	16
3	MusicXML	MusicXMLThe full MusicXML test-suite used to test the LilyPond program, available at: http://lilypond.org/doc/v2.17/↩ input/regression/	127	828	41760	6587	273
4	FictionBook	FictionBookSome randomly downloaded books available at: http://fictionbook-lib.org/	100	69379	3464352	650375	61
5	EPrintXML	EPrintXMLSome randomly downloaded descriptors form the Caltech Institute public repository, available at: http://caltechln.library.caltech.edu/↩ eprints/	50	3768	123081	26660	80
6	XHTML	XHTMLThe full version of the Koran published by LiberLiber and freely available at: http://www.liberliber.it/	125	5551	261855	33328	31
7	DocBook	DocBookSome randomly downloaded papers from the proceedings of the Balisage Series Conferences, available at: http://www.balisage.net/	117	3283	161337	61053	64
8	TEI	TEI Some randomly downloaded files from the Gutemberg Project available at: http://www.gutenberg.org/	90	40245	1965027	445865	92

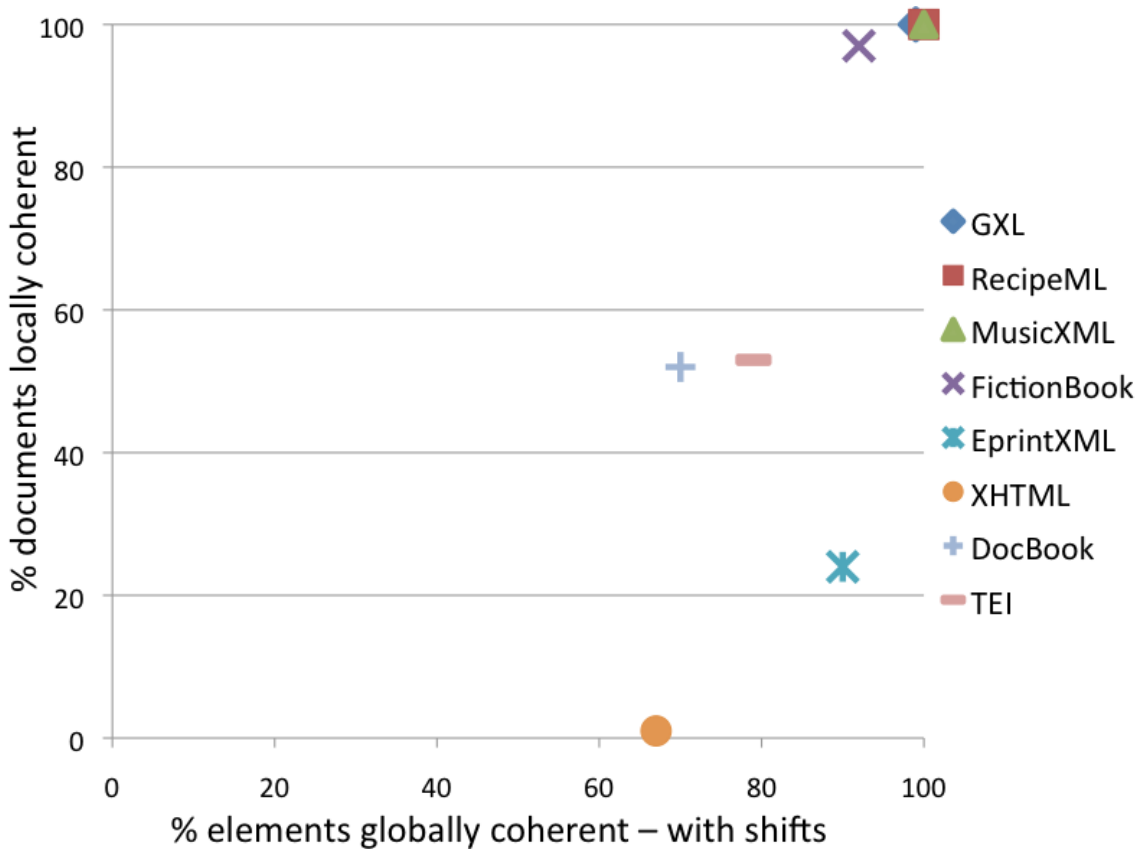


Figure 5.2: Figure 6. The percentage of locally coherent files and globally coherent elements for each language in the dataset.

are generated by a small number elements that impact on other elements. It is worth remarking that I did not analyse the definition of the elements in the vocabulary scheme (DTD or XML Schema or whatever), but rather I examined how these elements are actually used in real documents.

Thus, I organized this evaluation section in three parts: (i) vocabularies from which I managed to get global coherency, (ii) vocabularies from which such extraction was not possible but in which inconsistencies were localized and easy to spot and solve, and (iii) vocabularies whose usage is quite far from my model. In the next subsections I go into details of each vocabulary.

5.3.1 Full adherence or convergence to patterns

In three experiments I obtained global coherency. These vocabularies define the structure of graphs (GXL), musical scores (MusicXML) and ingredients of recipes (RecipeML). For each of them, in fact, it was possible to automatically derive a univocal patterns scheme by applying shifts. Table 5.2 summarizes these results.

Table 5.2: The result of checking patterns on three very structured vocabularies, which adhere to our theory natively or after a normalization phase.

	Set	#files	# locally coherent	# elements	# elements generating local incoherency	# elements globally coherent (no global shifts and containers generalization)	# elements globally coherent (with global shifts and containers generalization)
1	GXL	21	21 (100%)	14	0 (0%)	11 (80%)	14 (100%)
2	RecipeML	498	498 (100%)	16	0 (0%)	10 (62%)	16 (100%)
3	MusicXML	127	127 (100%)	273	0 (0%)	230 (84%)	273 (100%)

Before going into details of each vocabulary, it is interesting to discuss some commonalities among them. First of all, they are all data-centric. This means that the content is organized in highly regular structures, such as containers and records. Thus, the regular and rule-based approach suggested by my patterns fits very well the needs of the users. They also use few mixed content models, mainly for descriptions and comments, which reduces the number of shifts and makes it easier to assign patterns. The number of elements in the vocabulary, on the other hand, does not affect the results: GXT, using 14 elements, has a similar behaviour to MusicXML, which has 273.

- ***GXL***: 21 files using the Graph eXchanges Language, a format to describe graphs and define constructs such as edges, nodes and relations in a very

structured way. Each file in the dataset is locally coherent. 11 elements out of 14 are used in exactly the same way in all files (80% of the total elements in the data set). By applying global shifts and containers generalization, I easily managed to found a single pattern scheme valid for the whole dataset.

- ***RecipeML***: 498 documents, while the number of elements in the vocabulary is basically the same (16). All documents were locally coherent. Furthermore, 62% of the elements were assigned to the same pattern in all documents in the data set, up to 93% by applying global shift. This happened, for instance, to the element qty, used to indicate the amount of each ingredient in a recipe and recognized as *field* in 475 documents and as *meta* in 10 documents, and to the element title used 419 times as *field* and 79 times as *block*, and eventually classified as *block.*

- ***MusicXML***: 127 documents in a vocabulary that counts a much large number of elements than the others, 273. Yet, I managed to generate one globally coherent partition with the same techniques of the previous sets. 230 elements were associated with exactly the same pattern for all documents (84% of the elements), and the result were globally incoherent in a rather small number of documents, mainly due to incomplete data. Results were refined to 100% coherency by aggregating (1) different types of containers into a more general pattern (as in the case of the elements bend, type and para-list), or (2) empty fields and meta (as for fermata and part-name), or (3) blocks and fields (beats and key-alter), (4) elements recognized as *containers* in most of the documents, and in some cases, being empty, recognized as *meta* (that involves, for instance, the elements ornaments and rest). This happens because some information is not mandatory in the schema or simply missing in specific instances.

Considering that there are a few of these cases, I can shift into the pattern *Container* and achieve a reasonable global coherency on the whole set.

5.3.2 Large adherence

For two sets the number of elements that could not bring to local nor global coherency was low and I can conclude that they largely adhere to patterns. Table 5.3 summarizes my results.

Table 5.3: The result of checking patterns on some vocabularies, which adhere largely to our theory.

	Set	#files	# locally coherent	# elements	# elements generating local incoherency	# elements globally coherent (no global shifts and containers generalization)	# elements globally coherent (with global shifts and containers generalization)
4	FictionBook	100	97 (97%)	61	1 (2%)	34 (56%)	56 (92%)
5	EPrintXML	50	12 (24%)	80	2 (3%)	57 (71%)	72 (90%)

- ***FictionBook***: 100 documents compliant with FictionBook, an XML vocabulary to encode the structure of e-books, using a total of 61 elements. I can conclude that a large part of the schema substantially uses patterns. Three documents were locally incoherent: in most cases, the problem was with the element emph; the authors used often this element to emphasize entire paragraphs. In some cases the element was placed outside of the paragraph, in others just inside, around the textual content of the paragraph, in others around pieces of text, in others around inline elements (strong, sup). Such differences made impossible a straight interpretation of the element. Other troublesome cases are empty-line (recognized as *milestone* in 81 files and as *field* in 1 file, since it contained one whitespace character) and text-author (recognized as

atom in 36 files and as *table* in 1 file, since the content was structured in a sequence of very short paragraphs). For these 2 elements no shift or reduction was possible. I consider this an acceptable result.

- ***EprintXML*** The collection I studied was composed of 50 files encoding metadata about scientific papers, theses, reports and teaching material. The number of locally coherent files was very low: 12 over 50. This apparently rather bad result can be mitigated by observing that errors were connected to only two elements: type and url. In these files, in fact, these elements are recognized sometimes as *atoms* and sometimes as *fields,* thereby preventing any shift. Looking at data more carefully an interesting aspect comes to the light. Both these elements are direct children of the element item and are always used as *fields*. Everything would work if the element item was attributed to *record*. Unfortunately this element has been used in an odd way in a few bibliographic references, whose data was all specified as a plain text within one item, with one line for each entry and no internal structure. Such an odd choice made the algorithm recognize item as *block* and therefore type and url as *atoms*. The fact that there is no reachable coherency does not imply that the authors have preferred a different organization, but, in my mind, it is a side effect of the poor use of some elements. The other interesting point is that the errors on type and url impact only part of the dataset. In fact, I managed to assign one single pattern to 61 elements over 72 (90% of the total) by applying global shifts and containers generalization. The rest of the elements could not be restructured as patterns.

5.3.3 Partial adherence

The search of my patterns on some other vocabularies did not produce fully satisfactory results. That happened especially with languages that provide users several constructs and choices: the presence of content models that combine the same elements in very different ways, the nature of the languages that cover heterogeneous needs and narrower cases, the unconventional usage of some constructs make documents far from my pattern-based model. The results, discussed in detail in the following subsections, are summarized in Table 5.4.

Table 5.4: The result of checking patterns on some vocabularies, which adhere partially to my theory.

	Set	#files	# locally coherent	# elements	# elements generating local incoherency	# elements globally coherent (no global shifts and containers generalization)	# elements globally coherent (with global shifts and containers generalization)
6	XHTML	125	1 (1%)	31	2 (6%)	16 (51%)	21 (67%)
7	DocBook	117	62 (53%)	64	15 (23%)	9 (64%)	45 (70%)
8	TEI	90	48 (53%)	92	17 (18%)	27 (33%)	71 (79%)

- ***XHTML***: I collected 125 pages linked to each other and corresponding to different parts of the same book. The number of locally incoherent files was very high: 124 out of 125. Such inconsistencies depend on just 2 elements, that basically generated the same problem in all these files: table and a. The overall layout is organized through nested tables: some cells contain only logos and extra information, others contain menus and navigational buttons (that are again organized in tables), others just text content. Moreover, each page contains a navigation menu expressed as a table containing a elements to go back and forward and to access the table of content. The use of the

same tableanda elements for such a variety of purposes makes it impossible to assign them a single pattern and, as a consequence, overall results are distorted. Isolating these errors, I achieved good results. The elements can be split in three groups: 16 elements that were assigned the same pattern for all files (51%), 5 elements that can be reduced to a single pattern via global shifting and generalization (16%), and 10 elements (33%) that are problematic and confirm that the openness of the XHTML schema leads authors to create documents that are syntactically valid but, in my opinion, still unclear from a structural point of view.

- ***DocBook (Balisage)***: My experiment was on a collection of 117 DocBook files, for a total of 64 different elements. I found 55 locally incoherent documents (47% of the total). Differently from my previous experiments, several elements were involved in these local incoherences. In fact, there were 15 elements (23% of 64) that generated local incoherency, although only 5 of them were incoherent in more than 10 files. I established that most of the elements involved were locally incoherent because the Balisage DTD allows them to be used in more than a way. For instance, some authors used the element figure within a paragraph thus implicitly assigning it the *popup* role, while other times it has been used as direct child of containers and therefore recognised as yet another *container*. This variability should be interpreted neither as an error of the authors nor as a conflict between the pattern *popup* and *container*. Rather, it simply means that different authors used the same element in different ways. In particular, the element figure describes a precise structure according to its documentation, i.e. a block containing a display element (such as a mediaobject) and a title, which can be aligned to (typical of *con-*

tainers) or unaligned from (i.e. floating, typical of *popups*) the main flow of text. Global shifts and generalizations helped us to move towards coherency but this set is admittedly far from my pattern-based model. The 64% of the elements were actually given one single pattern even without reductions, while the final percentage was of 70%.

- ***Text Encoding Initiative***: The datasets included 90 files, using a total of 92 TEI elements. Half of the files (42) in the dataset were locally incoherent. The number of elements that generate incoherency in quite high (17), even if only 6 of them were incoherent in more than 10 files. Global shifts and containers generalizations improved these results (up to 33% and 79% of globally coherent files) but still achieved only partial adherence to patterns. My analysis on the TEI dataset produced results very similar to DocBook. I believe this is not a coincidence: the fact that they have to deal with very specific cases makes room for very different content models and contexts for the same elements in the vocabularies. As for DocBook, most of these problems derived from the ambivalent use of some elements. Truth is, these structures are valid and allowed by the TEI schema, so their different uses cannot be considered errors or misinterpretations.

There is an intrinsic opposition here between the minimality of my model and the richness and verbosity of these languages. On one side, this is not a problem since these two approaches are meant to cover different needs and have different applications. On the other, I believe that some simplification and re-structuring could also improve the readability and applicability of well-known vocabularies like TEI and DocBook.

Chapter 6

Leveraging structural patterns to build applications

The ability of patterns to capture the most relevant classes of structures and to express in a rigorous but simple way their relationships can be exploited to build novel tools for XML documents. For example, patterns allow us to build viewers that do not require users to directly master XML technologies but offer intuitive interfaces to read documents, move within their components, analyze their content and extract relevant information.

The crucial aspect is that these tools are independent of the document vocabularies since they work directly on their pattern-based representation. Since patterns can be extracted through automatic processing, these applications work on *any* document *without any knowledge of its original schema*. Thus, they are very helpful whenever the vocabulary is unknown, not available or available in a different version, and can help us to get an idea of the potential and applicability of the theory of patterns, and persuade us about the feasibility and quality of the pattern-based approach.

In this chapter I describe two tools I developed that are meant to support the ex-

ploration and analysis of heterogeneous document collections, and that work with no background information about the format documents are written in. The objective of the first tool, as described in Section 6.1, is to help the reader in navigating and exploring the document content. The pattern-based document analysis can also be used to perform more specific investigations: in Section 6.2, for example, I introduce another tool that supports the user in the task of searching the logical components (paragraphs, sections, titles, reference lists, bibliographic references, etc.) of scholarly documents. In order to demonstrate the effectiveness of this mechanism, I describe and evaluate an algorithm for the automatic identification of document components that has been developed using this tool.

6.1 Document Viewer

In this section I describe the Document Viewer[1], an interactive web-based tool aimed at supporting the reading, navigation and comprehension of documents. The document Viewer design is composed by two parts, as shown in Fig. 6.1 on the facing page: on the left side, a zoomable view based on the SunBurst technique [110] provides an overview of the whole document structure; on the right side, the content of the document is displayed in an hypertext-like fashion. The navigation of these two components is strictly coupled: for example, when a user hovers the mouse cursor over a text fragment in the viewer on the right, the corresponding element and all its ancestors are highlighted in the document hierarchy on the left; similarly, when a user focus on an element in the SunBurst view, the viewer scrolls to the corresponding text. This ability to display in a clear and coordinated way

[1]The Document Viewer is available online at the address http://eelst.cs.unibo.it/documentviewer/

both the content of the document and its global contexts is a key point of this tool.

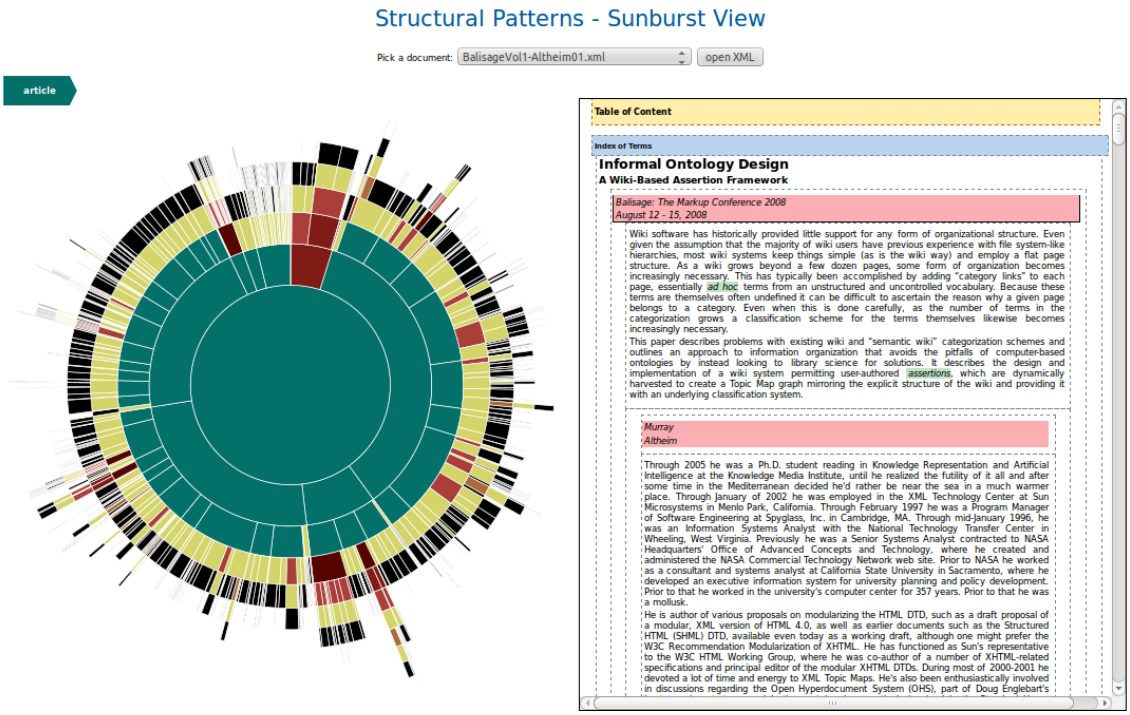


Figure 6.1: The layout of the Document Viewer

6.1.1 Conversion and generation of presentation rules

The information about the organization of the document expressed by structural patterns can be used to develop interfaces to read documents and explore their contents. I experimented with this approach in *Pviewer*, a subcomponent of Document Viewer that provides an hyper-text like representation of an XML document. It is a proof-of-concept Java and XSLT implementation that takes as input an XML document and produces an HTML page, plus some CSS and Javascript, with its content and structures. Fig. 6.2 on the next page, Fig. 6.3 on page 105 and Fig. 6.4 on page 108 show some zoom-in views of a possible output of PViewer generated

automatically from an XML file randomly chosen in our dataset.

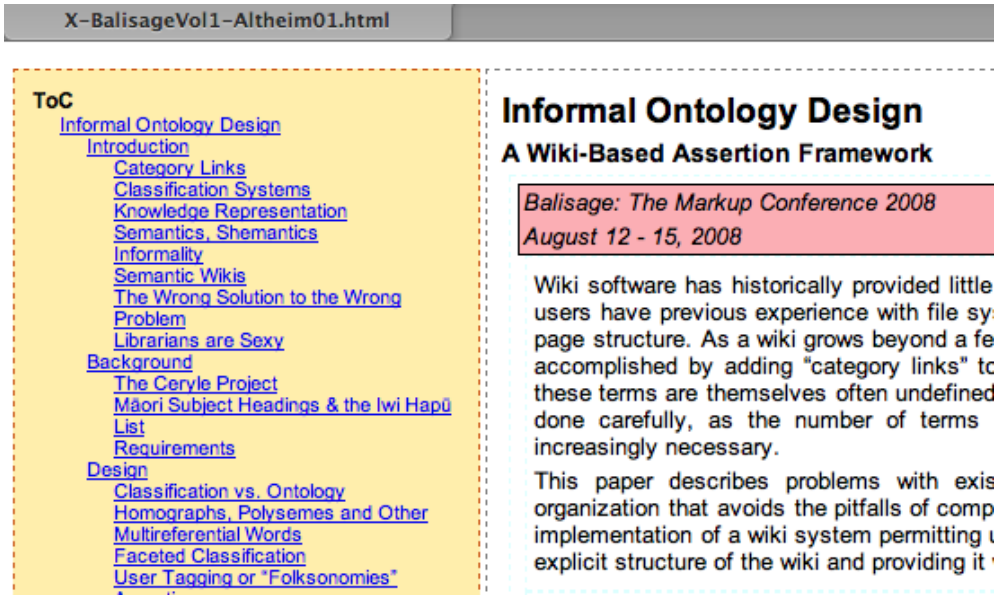


Figure 6.2: Basic visualization of a XML document in PViewer. The first blocks of the documents are shown in the right, beside an automatically-generated table of content.

Note that no XML tag is shown directly to the user but the page highlights the logical structures of the document: containers, blocks of text, text fragments, structured data and so on. The overall conversion process includes two steps, briefly described below.

Pattern identification

PViewer exploits the algorithm presented in Chapter 5 to identify patterns in any XML document. Two outputs are possible: in case of local coherency, the algorithm produces one single map where each element is associated with one pattern; if not, it assigns multiple patterns to some elements. In that case PViewer implements some reduction rules (basically, selecting the most general pattern within each sub-set) to produce a new map where each element is associated with only one pattern. This

makes the overall approach work also on documents that are not locally coherent, with acceptable results.

Conversion and generation of presentation rules

PViewer translates the original XML file into a HTML page composed of generic containers, blocks and inlines, associated with some CSS rules. Elements and presentation rules are generated from the map described above, and convey the structural meaning of each pattern. For instance, as shown in Fig. 6.3: containers are nested and shown with a border to clarify their containment relation, inline elements are highlighted with a darker background in contrast to plain text, milestones are replaced with images clicking on which users can read their XML source and attributes.

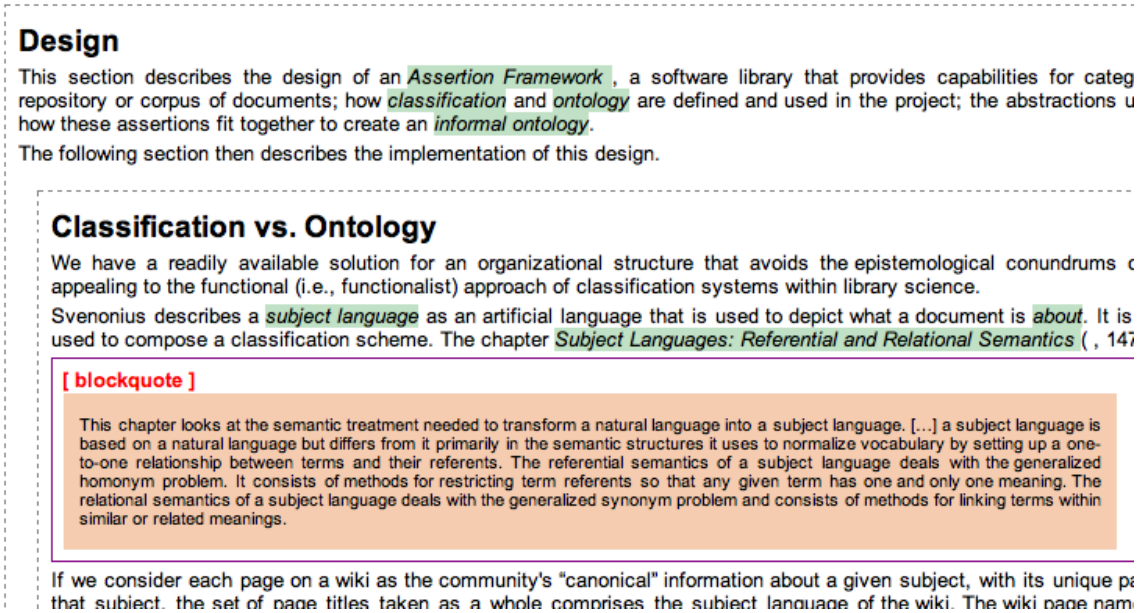


Figure 6.3: Details of visualization in PViewer: inlines use a darker background, and popups can be expanded on request. The hierarchical organization of containers is highlighted through dashed borders.

Besides showing how nested containers appear in PViewer, this image shows how it handles inlines and popups: the former use a darker background, while the latter

are displayed as boxes expanded on demand. The example is helpful to highlight a very important point: PViewer - and the overall theory of patterns - is not meant to capture peculiarities of each element in the vocabulary, rather to capture and show the basic logical structures in that vocabulary, *even without knowing it*. That is why all inline elements share the same presentation and there is no special formatting for specialized containers (for instance, abstract or bibliographies).

6.1.2 Information synthesis and extraction

Information extraction is the name given to any process that automatically extracts structured information from unstructured or semi-structured text. The main goal of this activity is to allow computation to be done on previously unstructured data. While early systems were based on handcrafted rule-based algorithms, most recent ones use machine learning algorithms starting from a collection of training examples. The current dominant techniques include Hidden Markov Models [12], Decision Trees [93], Maximum Entropy Models [18], Support Vector Machines [4] and Conditional Random Fields [75]. Another characteristic of information extraction systems is that the analysis has traditionally focused on satisfying precise, narrow, pre-specified requests from small homogeneous corpora and domains (e.g. biomedical datasets [94], news articles [75], informal text in emails [78], etc.)

In this section I present a different approach I used to extract relevant information from documents based on the pattern-based analysis. In particular, in PViewer I concentrate on the identification of two classes of information that support the navigation task: the table of contents and the index of terms. A key point of this approach is its generality: in fact, these operations are independent from the language in which the document is written in, require no background knowledge about

the document content (e.g. domain, context, genre, etc.), and may be performed without any previous information about the semantics and organization of the XML vocabulary.

The first element of the navigation interface generated by PViewer is table of content. As shown in the left part of the screenshot in Fig. 6.2 on page 104, it gives users a clear insight of the overall structure of the document and its hierarchical components. This table is created automatically from the data on headed containers, as titles are mapped into labels of the index, whose hierarchical positions reflect the order and nesting level within the XML document.

Patterns can also be exploited to extract a preliminary index of terms. In the pattern model, in fact, text fragments that carry a specific meaning within a flow of text are a*toms* or *inlines*. PViewer extracts all these fragments, removes some stop words and organizes them in alphabetical order. Fig. 6.4 on the following page shows a zoom-in view of the PViewer index of terms. The terms under letters 'A' and 'B' are visible in the image. The whole index is shown to the user when clicking on the 'Terms' button on the left.

There are several improvements possible for this component. For instance, I plan to add support for counting the number of occurrences of each term, filtering out some terms (for instance, by also integrating external linguistic components), linking terms to their occurrences in the text, aggregating statistical data, and so on. The pattern-based approach may be also combined with other information extraction techniques to improve the quality of the results: for example, the information about the function of text fragments in the document structure given by patterns, together their characterization in terms of document components (e.g. abstract, introduction, methods, problem statement, related work, etc. - see Section 6.2), may provide

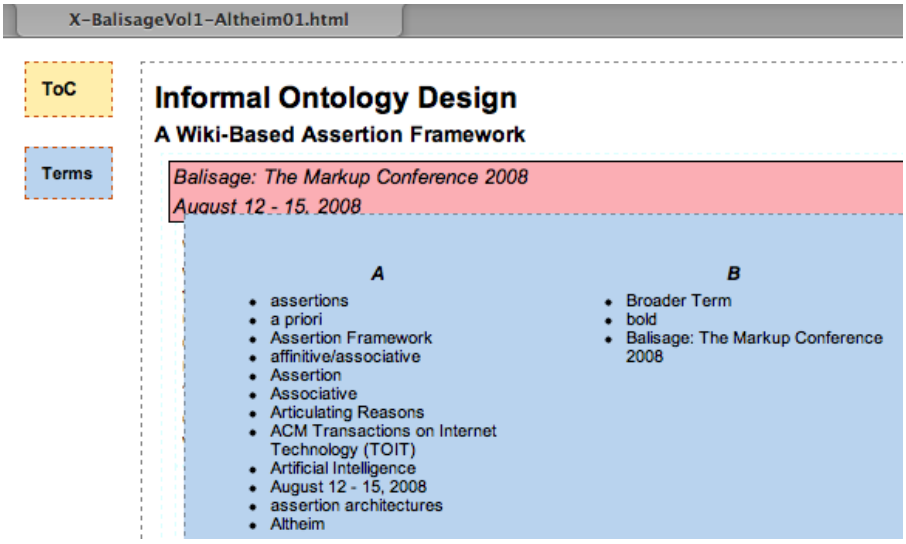


Figure 6.4: A zoom-in view of the basic index of terms generated by PViewer.

valuable hints to entity extraction methods based on NLP techniques. Another interesting feature is the interlinking of text documents with Linked Open Data [14]: for example, the approach developed in [77] to automatically annotate text documents with DBpedia [15] URIs may be used on the extracted terms, on the table of contents, and on some specific document components (e.g. abstract). This information can be exploited as background knowledge to implement search and faceted browsing functionalities.

6.1.3 Supporting reading, navigation and comprehension of documents

The last component of the Document Explorer introduces some elements borrowed from the visual analytics discipline in order to facilitate the navigation and exploration within documents. Visual analytics has been defined as "the science of analytical reasoning facilitated by interactive visual interfaces" [115], and is based on the idea of coupling human and machine analysis to support the process of inves-

tigation and sense making of huge amount of information. Although the objective of this discipline is to ultimately help users to make better decisions [68], some principles and techniques may also be leveraged in a document analysis context, as discussed in the rest of this section. For example, the ability to highlight patterns and trends about data can be used identify document components, as described in Section 6.2.2.

Design issues

The representation of the logical structure of a document may be reduced to the well-known problem of visualizing hierarchies. In order to make an efficient use of space, implicit tree visualizations (i.e. those that resort to an implicit representation of parent-child relations by positional encodings of the hierarchy items) must be preferred to explicit techniques (i.e. those that explicitly show parent-child relations as straight arcs or lines) [76]. In the last 30 years, a wealth of implicit visualizations have been proposed [92]. Among the possible alternatives, I decided to use the SunBurst technique [110], a space-filling visualization that uses a radial layout, as the base of the navigation view. In SunBurst, items in the document hierarchy are laid out radially, with the root element at the center and deeper levels farther away from the center. Colors are used to encode the structural role carry out by an element (i.e. the structural pattern of which the element is an instance of), and the angle swept out by each element corresponds to the number of characters contained, even recursively, by it.

This technique has been preferred to other well-known implicit tree visualizations[2]. A notable alternative is the Treemap [66][3], a space-filling slice-and-dice

[2]The interested reader can find a quite complete interactive catalog of tree visualization techniques at the address http://treevis.net. The outcomes of this work are summarized in [91].

[3]To be precise, a subclass of the Treemap named Ordered Treemap [98] should be used in order

technique based on a rectangular layout. In treemap, each element of a tree is depicted by a rectangle, which is then tiled with nested rectangles representing sub-branching. The color and area of each item correspond to attributes of the item as well: for example, these visual variables may be used to encode the pattern of the element and the number of contained characters, respectively. Comparing Treemap and Sunburst, the former has a longer learning curve and a less explicit portrayal of the hierarchy structure.

Interactive behaviours

One of the principles for browsing and searching at the basis of visual analytics is the so-called information-seeking mantra "overview first, zoom and filter, then details-on-demand" [97]. The basic idea is to develop interfaces that give a general context for understanding datasets by summarizing their most salient features (overview), reduce the complexity of the representation by removing extraneous information from the view and allowing for further data organization (zoom and filter), and finally reveal additional information on a point-by-point basis while the user interacts with the visualization (details-on-demand). As described in the rest of this section, this principle is the basis of the interactive behaviours in the Document Viewer.

The first information that is shown on user's demand concerns generic identifiers: when he/she hovers over an element, a serie of rectangles (with generic identifiers) are drawn in a box at the top of the SunBurst to represent the element and all its ancestors in the document hierarchy, ordered from left to right.

Another interactive behaviour is used to tackle the problem that, since the element size depends on the number of contained characters, the clarity of the hierarchy gradually degrades moving away from the root element. This problem has been

to preserve the order within the document hierarchy.

solved by adding the ability to zoom in and out: when the user clicks on an element, the SunBurst is reconfigured to show only the sub-tree rooted in that element (zoom in); to move up one step in the hierarchy, the user can click on the centre circle (zoom out).

Finally, interaction has also been used to keep the SunBurst view and the hypertext-like viewer coordinated during the user's investigation: in fact, when he/she hovers the mouse cursor over a text fragment in the viewer on the right, the corresponding element and all its ancestors are highlighted in the document hierarchy on the left; similarly, when a user focus on an element in the SunBurst view, the viewer scrolls to the corresponding text.

6.2 Document Component Extractor

In most disciplines, academic texts have established models of organisation and structure which are followed, more or less strictly, by all scholars and contributors. Some structures are shared across disciplines and capture very common objects of a text (such as tables, lists, references, front matter, etc.), others are specialised for specific disciplines (such as program listings in computer science works, epigraphs in humanities, medical histories in medicine, and so on).

Markup languages, and in particular XML vocabularies, provide authors with constructs to linearise these structural components. For instance, the element *para* in DocBook (a semantic markup language for technical documentation [121]), the element *p* of HTML [58], the element *block* of the legislative XML vocabulary called Akoma Ntoso [5], refer all to the same concept of one of a set of vertically-organised containers of (possibly styled) text often called a paragraph.

The idea at the base of this section is to shift my analysis of documents to a higher

level of abstraction, dealing with their structural components - such as paragraphs, lists, bibliographic references, sections, etc. - independently of the elements and the format of the markup language they are written in. In order to do this, I used a general, strong and shared conceptual model for the description of components, and exploited the pattern-based analysis to match the elements of each XML languages to it according to the best interpretation of their structural semantic roles. The result of this work is a pattern-based algorithm for the automatic recognition of such structural components.

The correct identification of logical components could provide many practical benefits, such as generating lists and summaries (including list of figures, tables, references and authors, tables of contents, etc.) automatically, enhancing the visualization of the content rendered in a Web browser window, and providing full-scale converters (or, in the worst case, robust stubs open to further development). The abstract representation of a document and its components can also be exploited to improve the comprehension of the document content, as remarked by [30], and build *Semantic Publishing* [100] [99] applications. Verifying semi-automatically some structural requirements of scientific papers, such as those expressed in [9] for the inclusion of XML-based vocabularies in PubMed Central, is a further possible application. Finally, on top of the identification of specific and inter-connected constructs – for instance all those structures related to bibliographic references like lists of references, inline citations, citation contexts – it will also be possible to implement sophisticated (cross-language) services for accessing, querying and manipulating such content.

The rest of the section is organized as follows. In Section 6.2.1 I give an overview of *DoCO* (the *Document Components Ontology*), the model that provides the general

structured vocabulary of document components I use in this chapter. In Section 6.2.2 I present the Document Component Extractor, a slightly modified version of the Document Viewer presented in Section 6.1 that I used to develop the algorithm for the automatic recognition of document components described in Section 6.2.3. Finally, in Section 6.2.4 I evaluate experimental results on real academic articles.

6.2.1 A model for document logical structure: DoCo

There exists an intrinsic complexity when defining some document components as purely rhetorical or purely structural. Let us consider as example a well-known component: the *paragraph*. A paragraph cannot be considered a pure structural component – i.e. a component carries only a syntactic function – since it *de facto* carries a meaning through its natural language sentences. Thus paragraphs have more than a syntactic attitude.

However, document markup languages such as HTML and DocBook define a paragraph as a pure structural component, without any reference to its rhetoric function:

- "A paragraph is typically a run of phrasing content that forms a block of text with one or more sentences" [58];
- "Paragraphs in DocBook may contain almost all inlines and most block elements" [121][4].

Here the term "block of text" and the verb "contains" emphasise the structural connotation of the paragraph, which is amplified by our direct experience as readers.

[4]The words *inline* and *block* in these list items do not refer to the structural pattern theory introduced previously, although some sort of overlapping exist.

Experience that implicitly tells us that a particular textual fragment shown in a book or in an HTML page is a paragraph rather than a chapter or a table.

The *Document Components Ontology* [39] (*DoCO*[5]), which I introduce in the rest of this section, has been developed so as to bring together the purely structural characterisation of document elements and their the purely rhetorical connotation. Besides including the *Pattern Ontology* (describing structural patterns)[6] described in Section 4.1 and *Discourse Element Ontology* (describing rhetorical components)[7], DoCO also defines other hybrid classes describing elements that are structural and rhetorical at the same time, such as *paragraph*, *section*, *list*, and the like.

Rhetorical foundations

A complete description of rhetorical components defined in DoCO is out of the scope of this work. However, it is useful to illustrate those that are actually used, in some way, to define the textual structures considered in my analysis, and that I introduce in the final part of this section.

All these pure rhetorical characterisations are defined in a particular ontology imported by DoCO, i.e. the *Discourse Element Ontology* (DEO), which provides a structured vocabulary for rhetorical elements within documents, enabling these to be described in RDF. The main class of this ontology is *DiscourseElement*, which describes all those elements of a document that carry out a rhetorical function. It is formally defined as follows[8]:

[5]DoCO, the Document Components Ontology: http://purl.org/spar/doco.
[6]PO, the Pattern Ontology: http://www.essepuntato.it/2008/12/pattern.
[7]DEO, the Discourse Element Ontology: http://purl.org/spar/deo.
[8]In this and the following excerpts I use the prefixes *po* to refer to entities defined in the Pattern Ontology, *deo* to refer to entities defined in the Discourse Element Ontology and *dcterms* that refers to entities defined in the Dublin Core Metadata Terms model [43]. Entities without prefixes are defined in the Document Components Ontology (DoCO).

deo:DiscourseElement $\sqsubseteq \top$

All the remaining rhetorical behaviours are modelled as subclasses of the above one. For the scope of this thesis, I introduce in detail only three of these classes: *Reference*, *Bibliographic Reference* and *Caption*.

A *reference* is a sort of link either to a specific part of the document or to another publication. In written text, small numbered superscripts standing for footnotes, items in a table of contents, items describing documents in a reference section of an article, can be modelled as individual of the class *Reference*, defined as follows:

deo:Reference $\sqsubseteq$ deo:DiscourseElement

Among all the possible kinds of references that can exists within a research article, recognising the bibliographic ones is a quite important issue to address, since they constitute the performative act of bibliographic citation. In particular, the class *BibliographicReference* describes references, usually contained in a footnote or a bibliographic reference list, that refer to another publication, such as a journal article, a book, a book chapter or a Web site. In DEO, it is defined as follows:

deo:BibliographicReference $\sqsubseteq$ deo:Reference

Textual structures in DoCO

All the aforementioned components are used as foundational blocks to define those classes of DoCO that bring together both a pure structural behavior (i.e. the structural patterns introduced before) and a generic rhetorical characterization (i.e. the rhetorical components briefly discussed at the beginning of this section). I particularly focus on those structures that usually define the main components of scientific

papers[9].

A *paragraph* is a self-contained unit of discourse that deals with a particular point or idea, structured in one or more sentences. In written text, the start of a paragraph is indicated by beginning on a new line, which may be indented or separated by a small vertical space by the preceding paragraph. In DoCO the class *Paragraph* is modelled as follows:

```
Paragraph ⊑
  deo:DiscourseElement ⊓
  po:Block ⊓
  ∃po:contains.Sentence
```

A *footnote* is a particular structure within a sentence that permits the author to make a comment or to cite another publication in support of the text, or both. A footnote is normally flagged by a superscript number immediately following that portion of the text to which it relates. For convenience of reading, the text of the footnote is usually printed at the bottom of the page or at the end of a text. In DoCO, the class *Footnote* is defined as follows:

```
Footnote ⊑
  deo:DiscourseElement ⊓
  (po:Container ⊔ po:Popup)
```

A *table* is a set of data arranged in cells within rows and columns. In XML file formats, it is usually organised in lines each containing a number of cells. From a pure structural pattern perspective, the element identifying the whole structure is organised according to the pattern *table* while those identifying the lines are always containers. The DoCO class *Table* is then defined as follows:

[9]Note that DoCO actually counts more classes that those described, that cover also other kinds of bibliographic entities, such as books.

```
Table ⊑
  deo:DiscourseElement ⊓
  po:Table ⊓
  ∃po:contains.po:Container
```

A *figure* is a communication object comprising one or more graphics, drawings, images, or other visual representations. In DoCO, it is modelled as a flat element without textual content, as introduced in the following excerpt:

```
Figure ⊑
  deo:DiscourseElement ⊓
  (po:Milestone ⊔ po:Meta)
```

Commonly, in scientific publications, both figures and tables are contained by captioned boxes (i.e. a *po:Container* containing a caption), which can be used to define a space within a document that contains either a figure (i.e. the class *FigureBox*) or a table (i.e. the class *TableBox*) and its caption, defined respectively as follows:

```
FigureBox ⊑
  CaptionedBox ⊓
  ∃dcterms:hasPart.Figure
TableBox ⊑
  CaptionedBox ⊓
  ∃po:contains.Table
```

A *list* is an enumeration of items, which may be composed by paragraphs, sequence of authors' names, etc. In DoCO, the class *List* is defined as follows:

```
List ⊑
  deo:DiscourseElement ⊓
```

```
po:Table ⊓
∃po:contains.po:Pattern ⊓
∀po:contains.(
  (po:Container ⊓
    ¬ (po:Table ⊔ po:HeadedContainer)) ⊔
  po:Field ⊔
  po:Block)
```

The above class is particularly useful for describing other, more specific, kinds of lists describing table of contents, list of figures, list of tables, and the like. In particular, the class *BibliographicReferenceLists* describes a list, usually within a bibliography, of all the references within the citing document that refer to journal articles, books, book chapters, Web sites or similar publications. It is defined in DoCO as follows:

```
BibliographicReferenceList ≡
  List ⊓
  ∀po:contains.deo:BibliographicReference
```

Of course, all these textual structures are usually contained in broader elements that aim at describing the overall organisation of the document structures. First, we have the *front matter*, i.e. the initial principal part of a document, usually containing self-referential metadata. Although in a book it can be quite massive, in a journal article the front matter is normally restricted to the title, authors and the authors' affiliation details, although the latter may alternatively be included in a footnote or the back matter. The DoCO class *FrontMatter* is defined as follows:

```
FrontMatter ⊑
  deo:DiscourseElement ⊓
```

```
  po:Container ⊓
  ∀po:isContainedBy.(¬ BodyMatter)
```

Along the line of the front matter, the *body matter* describes the central principal part of a document, that contains the real content. It may be subdivided hierarchically by the use of chapters and, as in research papers, sections. The class *BodyMatter* is disjoint to *FrontMatter*, since an element cannot be the initial and the central part of the document at the same time, and is defined as follows:

```
BodyMatter ⊑
  deo:DiscourseElement ⊓
  po:Container ⊓
  ∀po:isContainerBy.(¬ FrontMatter)
BodyMatter ⊓ FrontMatter ⊑ ⊥
```

The aforementioned elements can be composed by other textual structures used for a coarse-grained organisation of text, such as *sections*. The class *Section* describes entities used for a logical division of the text (organised in paragraphs), numbered and/or titled, which may contain subsections. It is defined in DoCO as follows:

```
Section ⊑
  deo:DiscourseElement ⊓
  po:HeadedContainer ⊓
  ∃po:contains.(Paragraph ⊔ Section)
```

Of course, in an article there exist particular kinds of sections that have a particular strucural and rhetorical function, such as the *bibliography*, i.e. that section containing a list of bibliographic references. In DoCO, the class *Bibliography* is defined as follows:

```
Bibliography ⊑
```

```
  Section ⊓
  ∃dcterms:hasPart.BibliographicReference
```

Sections and other high-level constructs such as chapters, captioned boxes or the document itself, can be introduced by a *title*. The DoCO class *Title* was introduced to describe a word, phrase or sentence that precedes and indicates the subject of a document or a document component. It is defined as follows:

```
Title ⊑
  deo:DiscourseElement ⊓
  (po:Block ⊔ po:Field) ⊓
  ∃po:isContainedByAsHeader.po:HeadedContainer
```

Starting from the above definition, it is then easy to describe particular kinds of titles, such as *section titles* modelled as the title being part of a particular section:

```
SectionTitle ⊑
  Title ⊓
  ∃po:isContainedByAsHeader.Section
```

6.2.2 An interactive tool for document component analysis

In this section I present the Document Component Extractor[10], a slightly modified version of the tool presented in Section 6.1 that I used to analyze documents and develop the algorithm for the automatic recognition of document components described in the next section. As all the algorithms and tools presented in this dissertation, the Document Component Extractor is language independent and works

[10]The Document Viewer is available online at the address http://eelst.cs.unibo.it/componentextractor/

on any XML document. The idea at the basis of my approach, in fact, is to leverage only the characterization provided by the pattern-based analysis to identify the structural roles (as defined in DoCo) of the content of documents stored in XML files. For example, a *doco:Paragraph* can be described as the block element (i.e. *po:*Block) with more occurrences in the document, a *doco:Section* can be defined as an headed container (i.e. *po:HeadedContainer*) that contains at least either one paragraph or one section, and a *doco:List* is a table (i.e. *po:Table*) that has all the child elements sharing the same name and pattern, which must be one out of the following ones: *po:Container*, *po:HeadedContainer*, *po:Record*, *po:Field* and *po:Block.* In order to check these hypothesis (and to formulate new ones) we need a language to define complex conditions, and a method to verify their validity. Instead of creating new languages and tools from scratch, I decided to use well-known technologies like Javascript and CSS.

For this purpose, I extended the Document Viewer with two textual areas, as shown if Fig. 6.5 on the next page: in the former the user can use Javascript (in particular JQuery[11]) code to specify the conditions, and to assign CSS classes to those elements that fulfill such conditions; in the latter the user can define CSS rules to specify a style for the classes assigned by the Javascript code. The previous informal characterizations for paragraphs, sections and lists can be easily converted into the following Javascript code[12]:

```
//DoCO-PARAGRAPHS:
var blockCount = {};
```

[11]http://jquery.com

[12]The results of the pattern-based analysis is used to assign CSS classes to the elements of the visualization. All these classes use the prefix "*po-*" followed by the name of the class of the Pattern Ontology to which they belong, as calculated by the algorithm described in Section 5.2. For example, the element *para* has been assigned to the class "*po-Block*", the element *section* to the class "*po-HeadedContainer*", etc.

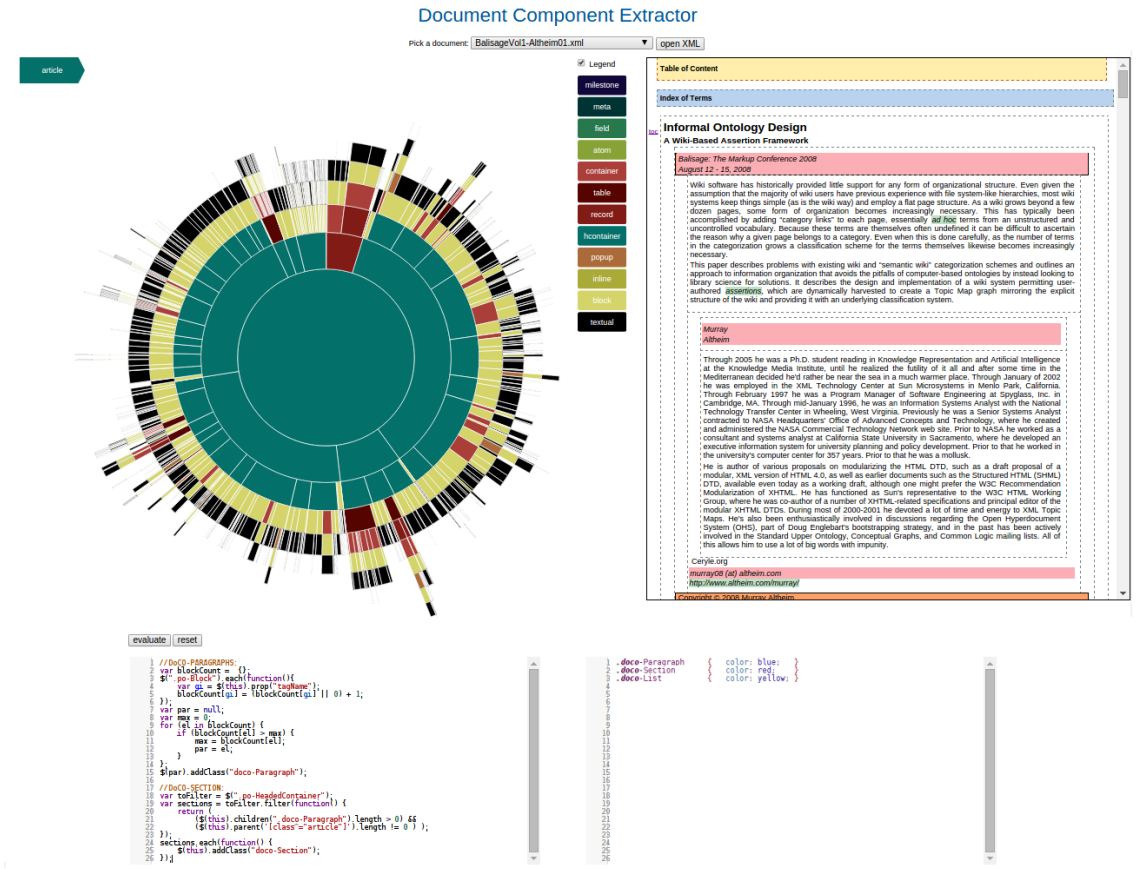


Figure 6.5: An overview of the Document Component Extractor

```
$(".po-Block").each(function(){
  var gi = $(this).prop("tagName");
  blockCount[gi] = (blockCount[gi] || 0) + 1;
});
var par = null;
var max = 0;
for (el in blockCount) {
  if (blockCount[el] > max) {
    max = blockCount[el];
    par = el;
```

```
  }
};
$(par).addClass("doco-Paragraph");

//DoCO-SECTION:
var toFilter = $(".po-HeadedContainer");
var sections = toFilter.filter(function() {
  return (
    ($(this).children(".doco-Paragraph").length > 0) &&
    ($(this).parent('[class^="article"]').length != 0 ) );
});
sections.each(function() {
  $(this).addClass("doco-Section");
});

//DoCO-LIST:
var toFilter = $(".po-Table");
var lists = toFilter.filter(function() {
  return (
    ($(this).children().length > 0) &&
    (
      $(this).hasClass("po-Container") ||
      $(this).hasClass("po-HeadedContainer") ||
      $(this).hasClass("po-Record") ||
      $(this).hasClass("po-Field") ||
      $(this).hasClass("po-Block")
    ) );
```

```
});
lists.each(function() {
  $(this).addClass("doco-List");
});
```

The next step consists in specifying the presentations rules that should be used to represent these new classes of elements:

```
.doco-Paragraph  {  color: blue;  }
.doco-Section    {  color: purple;  }
.doco-List       {  color: red;  }
```

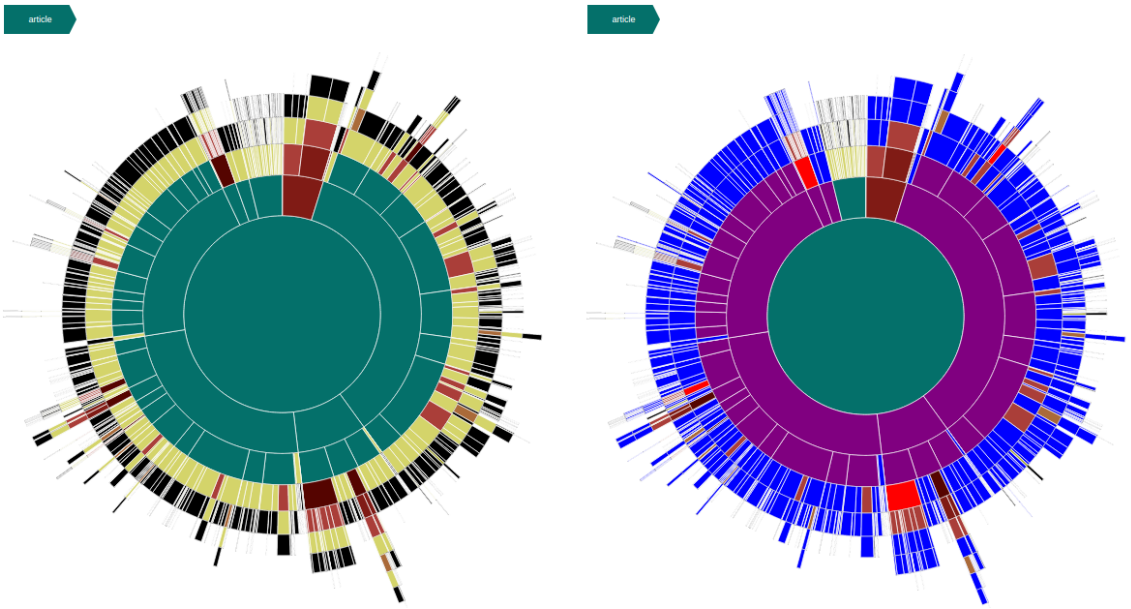


Figure 6.6: Document Component Extractor: an overview of a document before (on the left) and after (on the right) the execution of the Javascript and CSS codes.

Fig. 6.6 shows an overview of a document[13] before (on the left) and after (on the right) the execution of the Javascript and CSS codes. Elements recognized as *paragraphs* are blue, *sections* are purple and*lists* are red. Fig. 6.7 on the next page

[13]The XML file of the document used for all the examples in this section is available at the address http://balisage.net/Proceedings/vol1/html/Altheim01/BalisageVol1-Altheim01.html.

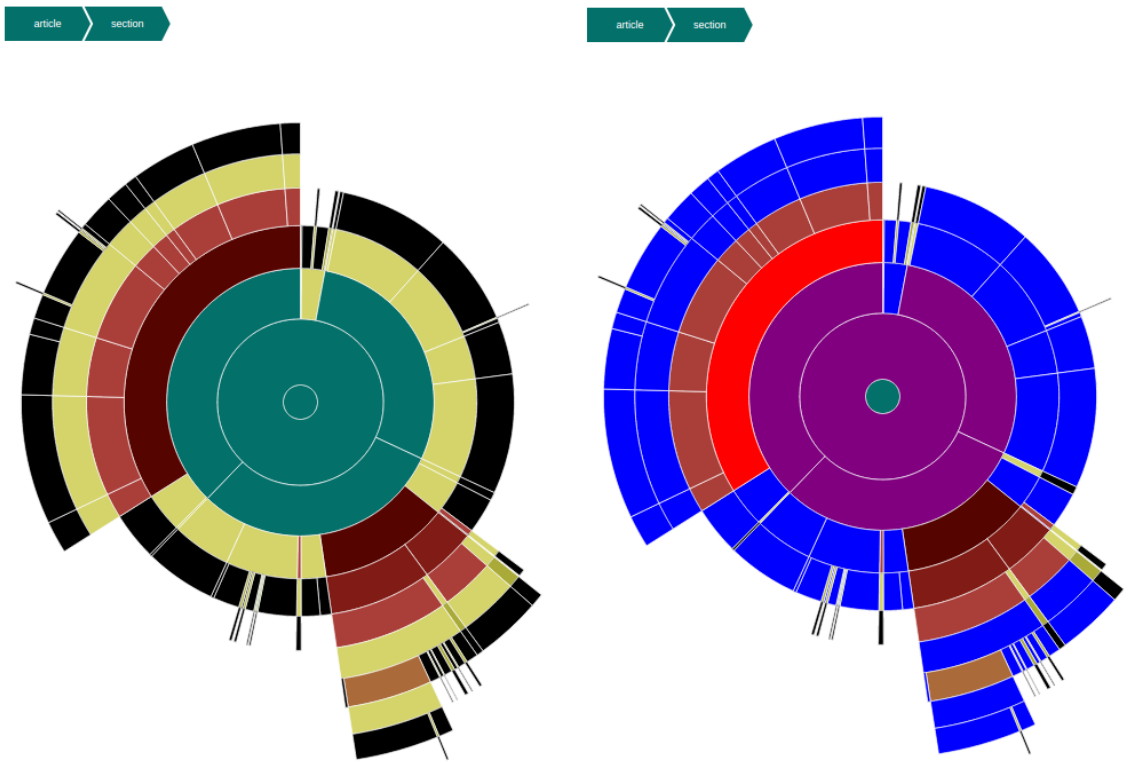


Figure 6.7: Document Component Extractor: a detail of a section composed of three subsections

depicts a detailed view of a section composed of three subsections, and Fig. 6.8 on the following page highlights the hypertext-like representation of the element recognized as *list*.

The user can browse the updated version of the SunBurst view and the hypertext-like visualization of the document to check the validity of his/her hypothesis, and to formulate new ones.

6.2.3 Recognizing document components in XML-based academic articles

In this section I introduce an algorithm I have developed that takes as input a set of XML sources of scientific articles that use the same vocabulary and recognizes

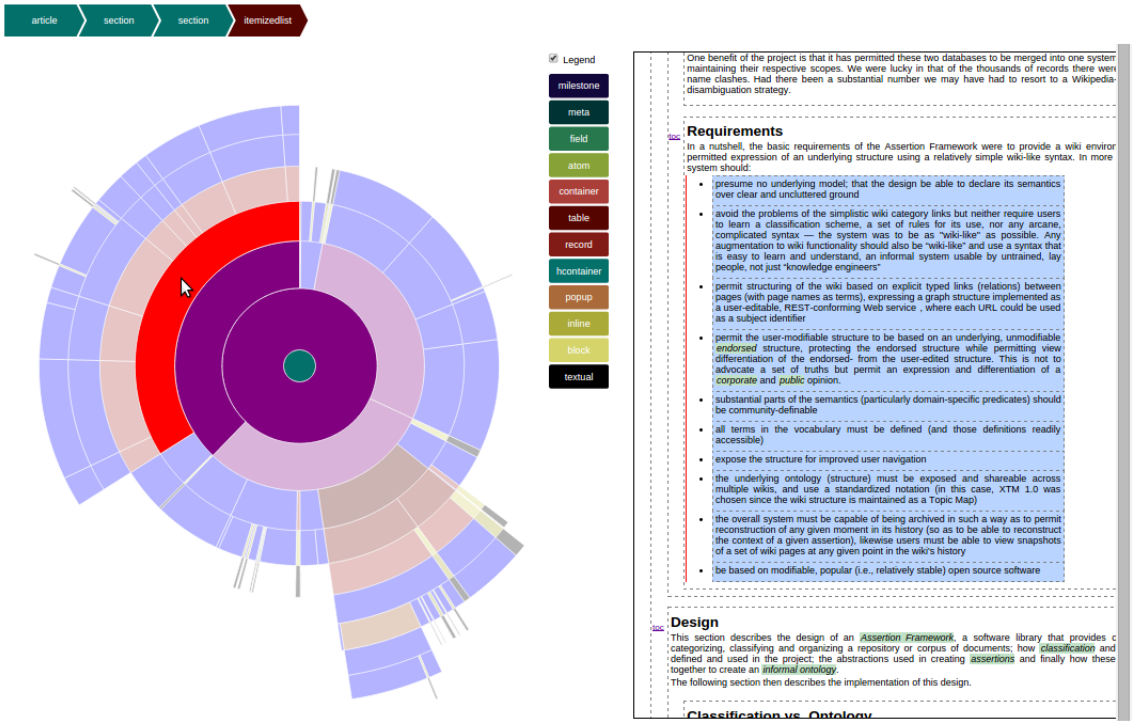


Figure 6.8: Document Component Extractor: the hypertext-like representation of the element recognized as *list*.

the document components defined in DoCO (see Section 6.2.1). The process is fully automatic: through three steps of analysis, my algorithm is able to associate the elements of the input documents with the corresponding classes in DoCO without relying on any background information about the vocabulary, its meaning, its intended scheme or the actual textual content of the documents themselves. It is worth noting that, although I believe that this approach may be applied to almost any type of documents, in this section I limit my preliminary analysis to academic articles.

During the first phase of this process the algorithm analyses separately every input document, and for each one performs the recognition of the structural patterns introduced in Section 4.1. To do this, I use the algorithm described in [38] for each document and produce a set of element-pattern bindings.

Since the way authors used to create the document according to a particular XML schema still allows one to use the same element in different structural ways (as a block, as a container, etc.), the analysis performed separately on each individual document can lead to the automatic assignment of different patterns to the same element (e.g. in one document the element *figure* is retrieved as popup, in another one as a container). Thus, in the second phase, the algorithm makes a synthesis of the results obtained in the previous step and assigns, in all the documents, the same pattern to a particular element: a pattern for all the elements named *para*, a pattern for all those named *section*, etc. The goal of this step is to identify these ambiguous situations and choose the pattern that best represents authors' use of each particular elements. In order to reduce the multiplicity of assignments, the algorithm considers every element that has been assigned to more than one pattern and proceeds as follows:

1. The algorithm applies a discrimination rule for containers, where it chooses a specific kind of container whenever an element was associated with more than one type of container (i.e. *po:Container*, *po:Table*, *po:Record* and *po:HeadedContainer*). Namely, if an element is associated with exactly two of these patterns, then it chooses the pattern that has the highest number of assignments (i.e. I apply the *"majority wins" rule*). Otherwise, if an element is associated with three or more of these patterns, then the element is assigned to *po:Container*, the most general case.

2. All the elements that, at this stage, are assigned to both the pattern *po:Container* (or its subclasses) and to the pattern *po:Popup* are always considered as of the former type only, regardless the majority wins rule.

3. I then apply a *pattern shift*: if element E is assigned to both pattern *P1* and

P2 and *P1* can be used in place of *P2*, then *E* has pattern *P1*. For instance, in my analysis the element *td* of Docbook (a cell in a table) can be recognised as both *po:Block* (including both text and elements) and *po:Field* (including only text) by the algorithm. Since a *po:Block* element can happen to just contain text and *po:Fields* can never contain other elements, then *td* elements can be assigned to the pattern *po:Block* without problems. In Fig. 6.9 I illustrate all the possible pattern shifts.

4. Finally, I applied the majority wins rule to perform discriminations in the remaining ambiguous scenarios.

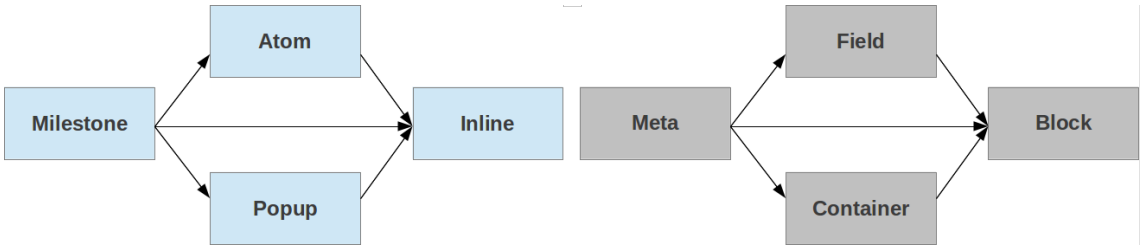


Figure 6.9: All the admissible shifts among patterns.

Thus, in the third phase, starting from the general element-pattern assignments resulting from the previous phase, I apply the following rules (in the order in which they are introduced) for each input document I took into account at the beginning of the process. The final result of this process will annotate the markup elements within the documents according to the textual structures defined in Section 6.2.1.

Paragraphs. Annotate with *Paragraph* all those markup elements that were annotated with pattern *co:Block* and that are the block element with more occurrences in the document.

Sections. Annotate with *Section* all those markup elements that contain at least either one paragraph or one section, that were annotated with pattern *po:HeadedContainer*,

and that are not the document element of the document.

Section titles. Annotate with *SectionTitle* all those markup elements that are header of (i.e. *po:isContanedByAsHeader*) a section.

Body matter. Annotate with *BodyMatter* all those markup elements that are not the document element, that were annotated with pattern *po:Container* (or any of its subclasses), that were not annotated with *Section* and are not contained (at any level) by sections, and that have as children the largest number of element annotated with *Section* (note: it must always contain one section at least). Since there can exist only one body matter within an article, in case multiple markup elements satisfy the previous rules, select, as body matter, the first of those elements taken according to a breadth-first visit of the markup document.

Front matter. Annotate with *FrontMatter* all those markup elements that are not the document element, that were annotated with pattern *po:Container* (or any of its subclasses), that were annotated neither with *Section* nor *BodyMatter*, that are not contained (at any level) by sections and body matters, and that have as children the smallest number of element annotated with *Section*. In addition, they must be placed before the body matter (if any). Since there can exist only one front matter within an article, in case multiple markup elements that satisfy the previous rules, select, as front matter, the first of those elements taken according to a breadth-first visit of the markup document.

Article title. Annotate with *Title* all those markup elements that were annotated with pattern *po:Field* or *po:Block* and that were not annotated with *Paragraph*. Since there can exist only one article title within an article, in case multiple markup elements that satisfy the previous rules, select, as title, the first of those elements taken according to a depth-first visit of the markup document.

Tables. Annotate with *Table* all those markup elements that contain at least two elements, that were not annotated with any of the aforementioned structures, that were annotated with pattern *po:Table*, that may have an element annotated with *po:Container* (or subclasses) as table header, and that have all the remaining child elements sharing the same name and pattern, which must be *po:Container* or any of its subclasses. In case of multiple descendant candidates, annotate with *Table* only the upper element.

Lists. Annotate with *List* all those markup elements that contain at least one other element, that were not already annotated as *Table*, that were annotated with structural pattern *po:Table*, and that have all the child elements sharing the same name and pattern, which must be one out of the following ones: *po:Container*, *po:HeadedContainer*, *po:Record*, *po:Field* and *po:Block*.

Figures. Annotate with *Figure* all those markup elements that were not previously annotated with any DoCO structure, that were annotated with structural pattern *po:Milestone* or *po:Meta*, and that have at least one attribute of which value is a valid URL ending with a file extension associated with an image file format.

Table boxes. Annotate with DoCO *TableBox* all those markup elements that were not previously annotated with any DoCO structure, that were annotated with structural pattern *po:Container* (or any of its subclasses but *po:Table*), and that contain at most three elements, of which at least one was annotated with *Table*. In case of multiple descendant candidates, annotate with *TableBox* only the upper element.

Figure boxes. Annotate with DoCO *FigureBox* all those markup elements that were not previously annotated with any DoCO structure, that were annotated with structural pattern *po:Container* (or any of its subclasses but *po:Table*) and contain

at most three elements, of which at least one is either a *Figure*, or a pattern *po:Block* containing only one element annotated with *Figure* and no text, or a *po:Container* (or its subclasses) containing (at any level) no textual blocks and an element annotated with *Figure*. In case of multiple descendant candidates, annotate with *FigureBox* only the upper element.

References. Annotate with *deo:Reference* all those markup elements that were annotated with pattern *po:Milestone*, that have an attribute *@x* with value equal or similar (i.e. the concatenation of "#" with the value) to the value of another attribute *@y* of another element (the name of these two attributes must differ). The latter element must be also linked by the reference element through the DCTerms property *dcterms:references*.

Bibliographic reference lists. Annotate with *BibliographicReferenceList* all those markup elements that were annotated with *List*, that hve all the children referenced by some reference. In case multiple elements satisfy the previous rules, consider as bibliographic reference lists only those that have at least one child referenced twice in the text.

Bibliography. Annotate with *Bibliography* all those markup elements that were annotated with *Section* and that contains either (a) an element annotated with *BibliographicReferenceList* or all the children but the section title referenced by some reference. In case multiple elements satisfy the previous rules, consider as bibliography only those that have at least a descendant referenced twice in the text.

Bibliographic references. Annotate with *deo:BibliographicReference* all those markup elements that are children of elements annotated with either *BibliographicReferenceList* or *Bibliography* (excluding section titles).

Footnotes. Annotate with *Footnote* all those markup elements that were not

annotated with any DoCO or DEO classes, and that were annotated either with pattern *po:Popup* or *po:Container*. In the former case, their closest ancestors annotated with *po:Block* must also be paragraphs, while the in latter case they must be referenced by an element annotated with *deo:Reference*.

6.2.4 Testing the algorithm

Some preliminary tests were performed to evaluate the effectiveness of my algorithm and its capability to identify the logical components of XML documents. In order to run tests I first implemented the algorithm in Java. My testing tool takes as input a collection of XML documents and annotates it with information about the structural role of each element in each document. The analysis – and the serialization of annotations – is not on XML but on the EARMARK representation [41] of the input documents. As described in Section 2.1.5, the basic idea of EARMARK is to model documents as collections of addressable text fragments, and to associate such text content with OWL assertions that describe structural features as well as semantic properties of that content. The EARMARK framework permits to add annotations in a fast, reliable and straightforward way.

I repeated the same experiment on two sets of documents, discussed separately in the following sections. For each of them, the process consisted of three steps[14].

Gold standard synthesis **:** I studied the vocabulary and assigned each of its elements to one or more DoCO structures. The analysis was subjective and solely based on their understanding of the semantics of the element, its definition schema and its documentation. On the other hand, I agreed on classifications that I consider reasonable. Notice that the same element could be associated with multiple DoCO

[14] All the materials and results of the experiments are available at http://www.essepuntato.it/2013/doco/test.

structures, that are all valid. Consider, for instance, the element `bibliography` of DocBook. It is clearly a *Bibliography* according to the DoCO ontology, but also a *Section* (a less specialized but equally correct characterization). Another point is important here: this mapping only includes elements that exist in the datasets and that experts agreed to associate with the DoCO structures I am interested in (see Section 6.2.3 for more details). All other elements are not taken into account. I do not aim at showing the completeness of DoCO in this work. Rather, I focus on a controlled set of elements in order to have a more precise evaluation of some preliminary extraction rules and DoCO constructs.

***DoCO mapping*:** the Java algorithm took as input the collection of documents and produced a map (encoded as RDF statements) that assigns each element to one or more DoCO structures. Some points are worth highlighting at this stage to better understand the following results. First of all, the fact that the algorithm assigns structures to each instance of each element in the documents. There is no effort to force one single assignment that holds for the whole vocabulary. It may well happen that the same element is used in two places according to two different DoCO structures, as noticed for the `bibliography` element. My goal it to check the quality of the algorithm on each instance, rather than to find a global classification. Notice also that the same element can be assigned to multiple structures: the two (or more) characterizations are meant to be both valid at the same time. It may not happen, for instance, that an element is characterized as either *Table* or *TableBox* since these two structures cover two different needs.

***Results comparison*:** the two sets of assignments were automatically compared. I measured their agreement in terms of *true positives* (TP), *false positives* (FP) and *false negatives* (FN) and I derived *precision P*, *recall R* and the *F1-score* to

get a more accurate view of the results. In particular, I calculated P as `TP/(TP+FP)`, R as `TP/(TP+FN)`, and consequently the F1 as `2*P*R/(P+R)`.

Let us now discuss separately how my algorithm performed on the two sets of documents I took into account.

Synthetic IML set

The first set is a collection of 18 IML documents, that use a total amount of 4764 XML elements. IML is a language, basically a subset of XHTML, fully based on my patterns. As expected, all elements of the language were correctly characterized, they being explicitly designed on my model (apart from an exception discussed below). This training experiment, in fact, was only performed to check basic functioning and minimum requirements of the algorithm on a controlled vocabulary and set of documents.

There was actually an interesting behaviour worth discussing. The algorithm found some false positives and negatives (175 elements) all ascribable to some dangling references in the text. The algorithm, in fact, characterizes as *bibliographicReferenceList* and *Bibliography* those sections that contain bibliographic references. These references must be linked by pointers within the same document (besides other constraints not relevant here, see section Section 6.2.3 for details). The presence of references that are not used but still in the document makes these elements to be incorrectly characterized. The error is then propagated to the characterization of their containers and sections. Nonetheless, such imperfection depends on the (wrong) encoding of the document rather than my extraction rules and algorithm.

Real DocBook set

The second set consists of all papers (i.e. 117 scientific papers) published in the Balisage Series Conferences, and primarily discuss research on document engineering and markup. There are several reasons for this choice: first of all, all the papers of the conference are freely available at `http://www.balisage.net` ; then, I know the community and the publication process, and I am personally certain that the authors of the papers are the actual authors of the XML versions available online (i.e., only a very limited editorial process affected the original XML documents); moreover, I know that the authors belong to a community composed of markup experts; finally, since the papers are encoded in the DocBook format, it is possible to consult the documentation and know the "correct analysis" of the data, thus obtaining a gold standard answer against which to compare the results of the algorithm.

These documents vary a lot in terms of internal structure and size: from 3K to 160K, with an average size of about 60K. Table 6.1 on the following page shows the map of associations (gold standard) against which I evaluated the algorithm's outcome. Note that not all elements of the vocabulary are listed, but only those I mapped to DoCO structures, and that the XPath expressions [parent::<element>] are used to indicate that the element is associated with that structure only if contained in a specific location.

The table shown in Fig. 6.10 on page 137 summarizes my comparison through the values of the parameters TP, FP, FN, precision, recall, F1-score as introduced before. The table also shows the elements associated with each DoCO structure and belonging to the set of TPs, FPs and Fns, useful for the following discussion. A point worth highlighting is that, even in presence of several false positives and negatives, they always involve a very small set of elements of the vocabulary.

Table 6.1: The assignment of each element of the DocBook schema in consideration to DoCO structures.

DoCO Structure	DocBook elements
Table	tbody, informaltable, variablelist
List	itemizedlist, orderedlist, keywordset
TableBox	table
Paragraph	para
Section	section, appendix, bibliography, abstract
SectionTitle	title[parent::section]
Figure	imagedata
FigureBox	figure, mediaobject[parent::para], mediaobject[parent::section], imageobject[parent::para], imageobject[parent::section]
BobyMatter	-
FrontMatter	info[parent::article]
Title	title[parent::article]
Reference	xref
BibliographicReferenceList	-
BibliographicReference	bibliomixed
Bibliography	bibliography
Footnote	footnote

The overall results, shown in the last row of the table, were very encouraging. The total values of precision and recall, in fact, are quite high (0.887 and 0.89).

It is interesting to discuss why some DoCO structures were recognized better than others. One of the reasons is that the declarations of the elements assigned to those structures in the DocBook vocabulary are more precise and stringent. The rules discussed in the previous section capture such behaviours and produced very good results. In other cases (for instance for *Tables* and *TableBox*) many more options are available to the users, some of which are not covered by the heuristics implemented in my algorithm. Let us discuss these issues in detail.

One clear result is that no element is assigned to the *bibliographicReferenceList* and *bodyMatter* DoCO classes. This is what I expected since no element belonging

Type	TP	TP elements	FP	FP elements	FN	FN elements	Prec.	Rec.	F1
bibliographic referencelist	0		0		0		-	-	-
paragraph	11562	para	2363	td bibliomixed	1036	para	0.830	0.917	0.871
figurebox	381	figure mediaobject	175	mediaobject imageobject listitem equation	282	figure	0.685	0.574	0.625
list	624	orderedlist itemizedlist keywordset	176	tr variablelist	107	orderedlist itemizedlist	0.78	0.853	0.815
table	135	informaltable variablelist tbody	62	orderedlist table itemizedlist figure thead	41	informaltable tbody variablelist	0.685	0.767	0.723
bibliography	49	bibliography	1	section	54	bibliography	0.98	0.475	0.640
section	1928	section appendix bibliography	0		117	abstract	1.0	0.942	0.970
reference	3408	xref	0		0		1.0	1.0	1.0
title	117	title	0		0		1.0	1.0	1.0
figure	564	imagedata	0		4	imagedata	1.0	0.992	0.996
bodymatter	0		0		0		-	-	-
sectiontitle	1928	title	0		0		1.0	1.0	1.0
frontmatter	117	info	0		0		1.0	1.0	1.0
footnote	392	footnote	24	listitem	59	footnote	0.942	0.869	0.904
bibliographic reference	1032	bibliomixed	2	section	965	bibliomixed	0.998	0.516	0.680
tablebox	52	table	13	figure	67	table	0.8	0.436	0.565
TOTAL	*22289*		*2816*		*2732*		*0.887*	*0.890*	*0.889*

Figure 6.10: The outcomes of the evaluation of the Balisage set.

to these classes had been identified in the preliminary human analysis, as shown in Table 6.1 on the facing page. The absence of false positives confirms that rules for such structures are accurate and reliable. There are other heuristics that worked very well on this dataset. There is in fact a complete match between the outcome of the algorithm and the assignments in the gold standard for other 4 DoCO structures (over 16, for a total of 25%): *Reference*, *Title*, *SectionTitle* e *FrontMatter*. Even in this case a precise and unambiguous characterization was possible.

The behaviour of the element `abstract` is worth discussing. It was expected to be recognized as a *Section* since it contains a sequence of blocks preceded by an optional title. In practice, authors did not include titles within abstracts but organized them in a plain sequence of text blocks. That explains the 117 false negatives for the *Section* class. Similar considerations can be applied to the elements `imagedata` that

are correctly recognized as *Figure* in 564 occurrences and in a very few cases (just 4) are missed. The heuristics, in fact, expect that element to be a milestone with an attribute pointing to a file of a given type (with a given extension). In these cases the extension is not supported and the results are not correct. The problem can be easily fixed by extending the set of supported filetypes.

The highest number of FPs and FNs was for the *Paragraph* DoCO structure. That is quite expected since paragraphs are the most common structures within scientific papers. What I did not expect was that so many `td` elements would be classified in that way, with more than 2000 FPs. There is a clear explanation for this if I look at the content of the elements. They all contain plain text, so that are mapped to the pattern *block* in the preliminary phase of the algorithm; in the second phase, being `td` the most used block in the document, all these elements are classified as DoCO *Paragraphs*. The issue here is in the practical use of the element `td`: it is often used as a container for blocks of text but there is no explicit element wrapping that block. Although I believe that is not a correct use of the element, I could refine and combine heuristics to also handle this specific case. The false negatives on the element `para` are connected to the same problem.

The results on the elements related to bibliographies are also very interesting. For structures *Bibliography* and *BibliographicReference*, in fact, the values of precision, recall and F1-score are considerably lower than other cases. There is a strong connection between these values. The *Bibliography* is in fact a special *Section* whose content is exclusively made of references, i.e. objects that are pointed by other elements in the text. The presence of blocks that are not recognized as references, or that do not contain references at all, makes the whole section to be classified in the wrong way. The solution to this problem will be to add a threshold that

indicates the percentage of pure references expected, so that even hybrid content can be classified in the same correct way.

The last specific case I would like to highlight is about *FigureBox* and *TableBox*. There are in fact several errors in recognizing these classes. The issue is again connected to a hybrid usage of the same element. In many cases, in fact, authors included `figure` elements to wrap tables. This element was classified as *FigureBox* by experts but as *TableBox* by the algorithm, that worked on its actual content. Similarly, the cases where the same element wraps plain text and formulas lead my algorithm to produce an unexpected classification.

The refinement of the heuristics should solve this issue and similar ones. The goal of this preliminary evaluation, in fact, was to pose the basis for further developments of my algorithm. I wanted to first identify which are the most common DoCO structures within real documents and to what extent they can be automatically recognized. Secondly, I wanted to identify the most relevant issues in order to refine current heuristics. Starting from the encouraging results I obtained from the tests, I plan to refine the heuristics I used in the algorithm so as to increase the precision and recall for each element in the golden standard. In addition, I also plan to extend the set of DoCO structures to identify automatically other significant ones, such as mathematical formulas, block-quotes and heading metadata (authors, affiliations, bios, etc.).

Chapter 7

Conclusions

The protagonist and main object of the analysis proposed in this thesis is the document, and in particular its digital representation. Over the years, very different communities, with heterogeneous objectives, skills and background have worked on the analysis of documents and, as a consequence, very different perspectives and interests lead scholars to stress on some aspect more than others: for example, semiologists focus on languages and signs, communication experts on message passing and immediacy, computer scientists on automatic analysis and transformations, psychologists on users' reactions and so on. Documents, in fact, are complex objects, and even defining what they really are and what they are used for is a complex issue.

A basic fact should be considered by all these different approaches: a document is the result of a writing process, made by an author, with the clear intent of storing and communicating information. It's no accident that the word root of the term "document" (derived from the Latin 'docere', that means 'teaching') focuses on such aspect: documents are means for constructing, progressing and disseminating ideas and data. Then documents (and in particular *digital* documents) cannot be conceived as indivisible units but they are the result of a complex process, where different and heterogeneous interventions work together to obtain the final output:

layered artifacts, where each layer is built for specific goals and that, combined together, create an effective unit of communication.

Among all these dimensions and layers, in this work I have focused on two of them, structure and content, that constitute the basic (and often indivisible) contribution written and organized by the authors. In particular I have analyzed the relations between content and structure in digital documents, trying to understand whether and to what extent an analysis based only on these aspects may be used to make sense of documents.

Starting from an analysis of real documents, I have identified regularities in how authors habitually organize their content, and derived classes of elements (patterns) with a precise structural characterization persisting across heterogeneous documents that use very different vocabularies. The outcome of this investigation is a novel approach to address document patterns, which captures the main constituents of the document organization by specifying a few meta-structures and some precise rules for combining them. This bottom-up approach has led to a formal definition of the model of structural patterns characterized by three main properties: *terseness* (a few objects and composition rules are sufficient to express the organization of documents), *coverage* (any possible situation in any document is captured by the model) and *expressiveness* (patterns make explicit the semantics of structures, relations and dependencies).

The main contribution of this work is then providing a theory that is able to represent and capture the model used by authors of documents to organize their contents and communicate their messages. Working on different vocabularies, contexts, domains and communities, the theory has proved to be general, effective and robust. A key aspect of this work, in fact, is the ability to identify structural patterns in

an automatic way with no background information about vocabularies, schemas, or their intended meaning. The logic behind the theory of patterns guarantees that no trivial configuration is ever used either at the local level, that is, for the markup elements within a particular document, or at the global level, for all the elements of a larger set of documents in the same XML vocabulary. For instance, it is impossible to assign the same pattern, for example, Inline, to the whole set of elements being examined, since this would result in making the ontological characterization of patterns described totally inconsistent. In addition, the shift rules and the container generalization mechanism introduced (both in locally and globally defined scenarios) guarantee the identification of the most meaningful choice of patterns in terms of granularity: the algorithm always retrieves the most specific pattern for an element given its structure in terms of context and content model. This guarantees that the largest possible set of patterns will be considered and, if possible, selected.

The adoption of a pattern-based approach has many practical benefits in all stages of the documents' lifecycle: they can be used as *indicators* to compare how different communities organize their discourse and study to what extent their documents share design principles and guidelines, *guidelines* for creating well-engineered documents and vocabularies, rules for extracting structural components and other useful information from legacy documents, etc.

As regards the first point, the algorithm for the automatic identification of structural patterns has already proved to provide useful information about documents that can be used for retrospective analysis. An important research direction I want to carry on is investigating whether there are differences in the way in which different authors actually use certain grammars, and to what extent they share design rules. I want also to compare the production of heterogeneous communities of authors

(e.g., from different disciplines, with different backgrounds, etc.) that use the same language, in order to further verify the validity and the adequacy of the pattern theory. Moreover, I plan to test the pattern theory on documents with overlapping situations , in order to investigate its applicability, and to explore if it is able to provide some useful information about such complex contexts.

Another useful application of patterns concerns document engineering and the possibility to perform semiautomatic refactoring operations. For example, I want to study the relation between the organization of the content and the overall quality of the document, and investigate how the internal structure of documents, the formal definition of schemas, and the suggested community guidelines concerning the use of a particular schema can be improved by adopting patterns.

Finally, applications, services and tools that can be developed using structural patterns are the last (and probably most important) aspect that should be taken into consideration in order to understand and evaluate this work. As shown in the last part of the thesis, patterns can be used to build hypertext-like visualizers, develop browsers that support the exploration of the document content, extract tables of contents and indexes of terms. But the most interesting perspective opened by this approach is the possibility of using structural patterns as the foundations for further analysis aimed at (re)constructing the different layers in which the information contained in the document is organized. For example, I presented the Document Component Extractor, a tool I developed that supports the identification of higher level information such as document components (e.g. abstract, introduction, methods, problem statement, related work, etc.) in scholarly articles solely on the basis of the structural information provided by patterns. Another pattern-based application I'm developing concerns the identification, extraction and characterization of

citation networks emerging from document collections. As future work, I also plan to use patterns to support the extraction of information about two additional levels: rhetoric and argumentative.

A final remark concerns Semantic Web Technologies, which has proved to provide an effective environment for expressing semantic-enriched machine-readable representations of documents and related models, in which the information about documents can be expressed in a clear and explicit manner, and can thus be easily retrieved, managed, provided to the user and used for further computations and analysis.

References

[1] Alexander, C. (1979). The Timeless Way of Building. Oxford University Press, 1979.

[2] Alexander, C., Ishikawa, S., Silverstein, M., 1977. A pattern language: towns, buildings, construction (Vol. 2). Oxford University Press.

[3] André, J. (1998). Petite histoire des signes de correction typographique. Cahiers GUTenberg, (31), 45-59.

[4] Asahara, M., Matsumoto, Y. (2003, May). Japanese named entity extraction with redundant morphological analysis. In Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology-Volume 1 (pp. 8-15). Association for Computational Linguistics.

[5] Barabucci, G., Cervone, L., Palmirani, M., Peroni, S., Vitali, F. (2009). Multi-layer markup and ontological structures in Akoma Ntoso. In Casanovas, P., Pagallo, U., Sartor, G., Ajani, G. (Eds.), Proceeding of the International Workshop on AI approaches to the complexity of legal systems II (AICOL-II). Berlin, Germany: Springer.

[6] Barabucci, G., Di Iorio, A., Peroni, S., Poggi, F., Vitali, F. (2013). Anno-

tations with EARMARK in practice: a fairy tale. In F. Tomasi, F. Vitali (Eds.), Proceedings of the 2013 Workshop on Collaborative Annotations in Shared Environments: metadata, vocabularies and techniques in the Digital Humanities (DH-CASE 2013). New York, New York, US: ACM Press. DOI: 10.1145/2517978.2517990

[7] Barabucci, G., Peroni, S., Poggi, F., Vitali, F. (2012). Embedding semantic annotations within texts: the FRETTA approach. In Proceedings of the 2012 ACM Symposium on Applied Computing (SAC 2012): 658–663. New York, New York, US: ACM Press. DOI: 10.1145/2245276.2245403

[8] Barnard, D., Hayter, R., Karababa, M., Logan, G., McFadden, J. (1988). SGML-based markup for literary texts: Two problems and some solutions. Computers and the Humanities, 22(4), 265-276. doi:10.1007/BF00118602.

[9] Beck, J. (2010). Report from the Field: PubMed Central, an XML-based Archive of Life Sciences Journal Articles. In Proceedings of the International Symposium on XML for the Long Haul: Issues in the Long-term Preservation of XML. DOI: 10.4242/BalisageVol6.Beck01

[10] Berglund, A., Boag, S., Chamberlin, D., Fernández, M. F., Kay, M., Robie, J., Siméon, J. (2010). XML Path Language (XPath) 2.0 (Second Edition). W3C Recommendation 14 December 2010. World Wide Web Consortium. http://www.w3.org/TR/xpath20/ (last visited November 26, 2014).

[11] Bikakis, N., Tsinaraki, C., Gioldasis, N., Stavrakantonakis, I., Christodoulakis, S. (2013). The XML and Semantic Web Worlds: Technologies, Interoperability and Integration: A Survey of the State of the Art. In Semantic Hyper/Multimedia Adaptation (pp. 319-360). Springer Berlin Heidelberg.

[12] Bikel, D. M., Miller, S., Schwartz, R., Weischedel, R. (1997, March). Nymble: a high-performance learning name-finder. In Proceedings of the fifth conference on Applied natural language processing (pp. 194-201). Association for Computational Linguistics.

[13] Bischof, S., Decker, S., Krennwallner, T., Lopes, N., Polleres, A. (2012). Mapping between RDF and XML with XSPARQL. Journal on Data Semantics, 1(3), 147-185.

[14] Bizer, C., Heath, T., Berners-Lee, T. (2009). Linked data-the story so far. International journal on semantic web and information systems, 5(3), 1-22.

[15] Bizer, C., Lehmann, J., Kobilarov, G., Auer, S., Becker, C., Cyganiak, R., Hellmann, S. (2009). DBpedia-A crystallization point for the Web of Data. Web Semantics: science, services and agents on the world wide web, 7(3), 154-165.

[16] Blomqvist, E., Sandkuhl, K. (2005, May). Patterns in Ontology Engineering: Classification of Ontology Patterns. In ICEIS (3) (pp. 413-416).

[17] Boag, S., Chamberlin, D., Fernández, M. F., Florescu, D., Robie, J., Siméon, J. (2010). XQuery 1.0: An XML Query Language (Second Edition). W3C Recommendation 14 December 2010. World Wide Web Consortium. http://www.w3.org/TR/xquery/ (last visited November 26, 2014).

[18] Borthwick, A., Sterling, J., Agichtein, E., Grishman, R. (1998). NYU: Description of the MENE named entity system as used in MUC-7. In In Proceedings of the Seventh Message Understanding Conference (MUC-7.

[19] Burget, R. (2005). Visual HTML document modeling for information extraction. In Proceedings of the 2005 International Workshop on Representation

and Analysis of Web Space (RAWS 2005): 17-24. http://ftp.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-164/raws2005-paper2.pdf (last visited November 26, 2014)

[20] Cardoso, E., Jabour, I., Laber, E., Rodrigues, R., Cardoso, P. (2011). An efficient language-independent method to extract content from news webpages. In Proceedings of the 11th ACM symposium on Document engineering (DocEng 11): 121-128. DOI: 10.1145/2034691.2034720

[21] Chidlovskii, B. (2001). Schema Extraction from XML: A Grammatical Inference Approach. In KRDB (Vol. 45).

[22] Chung, C. Y., Gertz, M., Sundaresan, N. (2002). Reverse engineering for web data: From visual to semantic structures. In Proceedings of the 18th International Conference on Data Engineering (ICDE 02): 53-63. DOI: 10.1109/ICDE.2002.994697

[23] Ciccarese, P., Groza, T. (2011). Ontology of Rhetorical Blocks (ORB). Editor's Draft, 5 June 2011. World Wide Web Consortium. http://www.w3.org/2001/sw/hcls/notes/orb/ (last visited December 12, 2014).

[24] Ciccarese, P., Peroni, S. (2013). The Collections Ontology: creating and handling collections in OWL 2 DL frameworks. Semantic Web – Interoperability, Usability, Applicability. DOI: 10.3233/SW-130121

[25] Clark, J. (2001). RELAX NG Specification. Committee Specification. Organization for the Advancement of Structured Information Standards. http://relaxng.org/spec-20011203.html (last visited November 26, 2014).

[26] Colazzo, D., Sartiani, C., Albano, A., Manghi, P., Ghelli, G., Lini, L., Paoli, M. (2002). A typed text retrieval query language for XML documents. In Journal of the American Society for Information Science and Technology, 53 (6): 467-488. DOI: 10.1002/asi.10059.

[27] Coombs, J. H.,Renear A. H., DeRose, S. J. (1987). Markup Systems and the Future of Scholarly Text Processing. Communications of the ACM 30 (11): 933-947. DOI: 10.1145/32206.32209.

[28] Dattolo, A., Di Iorio, A., Duca, S., Feliziani, A.A., Vitali, F. (2007). Structural patterns for descriptive documents. In Baresi, L., Fraternali, P., Houben, G. (Eds.), Proceedings of the 7th International Conference on Web Engineering 2007 (ICWE 2007). Berlin, Germany: Springer. DOI: 10.1007/978-3-540-73597-7_35

[29] De La Higuera, C. (2005). A bibliographical study of grammatical inference. Pattern recognition, 38(9), 1332-1348.

[30] De Waard, A. (2010). From Proteins to Fairytales: Directions in Semantic Publishing. In IEEE Intelligent Systems, 25 (2): 83-88. DOI: 10.1109/MIS.2010.49.

[31] De Waard, A. (2010). Medium-Grained Document Structure. http://www.w3.org/wiki/HCLSIG/SWANSIOC/Actions/RhetoricalStructure/models/medium (last visited November 26, 2014).

[32] Della Penna, G., Magazzeni, D., Orefice. D. (2010). Visual extraction of information from web pages. In Journal of Visual Languages and Computing, 21 (1): 23-32. DOI: 10.1016/j.jvlc.2009.06.001

[33] DeRose, S. (2004). Markup Overlap: A Review and a Horse. In Proceedings of the Extreme Markup Languages 2004. Rockville, MD, USA: Mulberry Technologies, Inc. http://conferences.idealliance.org/extreme/html/2004/DeRose01/EML2004DeRose01.html (last visited November 26, 2014).

[34] DeRose, S. J. , Durand, D., Mylonas, E., Renear, A. H. (1990). What is Text, Really? In Journal of Computing in Higher Education, 1 (2), 3–26, 1990.

[35] DeRose, S., Maler, E., Daniel, R. (2001). XPointer xpointer() Scheme. W3C Working Draft, 19 December 2002. World Wide Web Consortium. http://www.w3.org/TR/xptr-xpointer/ (last visited November 26, 2014).

[36] Di Iorio, A. (2007). Pattern-based segmentation of digital documents: model and implementation. PhD Thesis. University of Bologna.

[37] Di Iorio, A., Gubellini, D., Vitali, F. (2005). Design patterns for document substructures. In Proceedings of the Extreme Markup Languages 2005. Rockville, MD, USA: Mulberry Technologies, Inc. http://conferences.idealliance.org/extreme/html/2005/Vitali01/EML2005Vitali01.html (last visited November 26, 2013).

[38] Di Iorio, A., Peroni, S., Poggi, F., Vitali, F. (2012). A first approach to the automatic recognition of structural patterns in XML documents. In Proceedings of the 2012 ACM symposium on Document Engineering (DocEng 2012), New York, ACM, 2012, 85-94. DOI: 10.1145/2361354.2361374

[39] Di Iorio, A., Peroni, S., Poggi, F., Vitali, F., Shotton, D. (2013). Recognising document components in XML-based academic articles. In Proceedings of the

2013 ACM symposium on Document Engineering (DocEng 2013): 181-184. New York, New York, USA: ACM. DOI: 10.1145/2494266.2494319

[40] Di Iorio, A., Peroni, S., Vitali, F. (2009). Towards markup support for full GODDAGs and beyond: the EARMARK approach. In Proceedings of Balisage: The Markup Conference 2009. Rockville, Maryland, USA: Mulberry Technologies, Inc. DOI: 10.4242/BalisageVol3.Peroni01. http://balisage.net/Proceedings/vol3/html/Peroni01/BalisageVol3-Peroni01.html (last visited November 26, 2014).

[41] Di Iorio, A., Peroni, S., Vitali, F. (2011). A Semantic Web approach to everyday overlapping markup. Journal of the American Society for Information Science and Technology, 62(9): 1696–1716. DOI: 10.1002/asi.21591

[42] Di Iorio, A., Peroni, S., Vitali, F. (2011). Using semantic web technologies for analysis and validation of structural markup. International Journal of Web Engineering and Technology, 6(4): 375–398. DOI: 10.1504/IJWET.2011.043439

[43] Dublin Core Metadata Initiative (2010). DCMI Metadata Terms. DCMI Recommendation. http://dublincore.org/documents/dcmi-terms/ (last visited November 26, 2014).

[44] Durusau, P., O'Donnell, M. B. (2002). Just-In-Time-Trees (JITTs): Next Step in the Evolution of Markup. In Proceedings of 2002 Extreme Markup Languages Conference, Montréal, Canada.

[45] Falco, R., Gangemi, A., Peroni, S., Vitali, F. (2014). Modelling OWL ontologies with Graffoo. In ESWC 2014 Satellite Events - Revised Selected Papers, Lecture Notes in Computer Science. Berlin, Germany: Springer. Post-

print available at http://speroni.web.cs.unibo.it/publications/falco-in-press-modelling-ontologies-graffoo.pdf (last visited November 26, 2014)

[46] Gamma, E., Helm, R., Johnson, R., Vlissides, J. (1994). Design patterns: elements of reusable object-oriented software. Addison-Wesley, New York, 1994.

[47] Gangemi, A., Presutti, V. (2009). Ontology design patterns. In Handbook on Ontologies (pp. 221-243). Springer Berlin Heidelberg.

[48] Gao, S., Sperberg-McQueen, C. M., Thompson, H. S. (2012). W3C XML Schema Definition Language (XSD) 1.1 Part 1: Structures. W3C Recommendation 5 April 2012. World Wide Web Consortium. http://www.w3.org/TR/xmlschema11-1/ (last visited November 26, 2014).

[49] Georg, G., Hernault, H., Cavazza, M., Prendinger, H., Ishizuka, M. (2009). From Rhetorical Structures to Document Structure: Shallow Pragmatic Analysis for Document Engineering. In Proceedings of the 2009 ACM symposium on Document engineering (DocEng09). New York, New York: ACM. DOI:10.1145/1600193.1600235.

[50] Georg, G., Jaulent, M. (2007). A Document Engineering Environment for Clinical Guidelines. In Proceeding of the 2007 ACM symposium on Document engineering (DocEng07). New York, New York: ACM. DOI:10.1145/1284420.1284440.

[51] Goldfarb, C. F. (1981). A generalized approach to document markup. In ACM Sigplan Notices (Vol. 16, No. 6, pp. 68-73). ACM.

[52] Goldfarb, C. F. (1990). The SGML Handbook. New York, New York, USA: Oxford University Press. ISBN: 0198537373.

[53] Groza, T., Handschuh, S., Möller, K., Decker, S. (2007). SALT – Semantically Annotated LaTeX for Scientific Publications. In Franconi, E., Kifer, M., May, W. (Eds.), Proceedings of the fourth European Semantic Web Conference (ESWC 2007). Berlin, Germany: Springer.

[54] Groza, T., Möller, K., Handschuh, S., Trif, D., Decker, S. (2007). SALT: Weaving the claim web. In Aberer, K., Choi, K., Noy, N. F., Allemang, D., Lee, K., Nixon, L. J. B., Golbeck, J., Mika, P., Maynard, D., Mizoguchi, R., Schreiber, G., Cudré-Mauroux, P. (Eds.), Proceedings of 6th International Semantic Web Conference and of the 2nd Asian Semantic Web Conference (ISWC 2007 + ASWC 2007). Berlin, Germany: Springer.

[55] Groza, T., Möller, K., Handschuh, S., Trif, D., Decker, S. (2007). SALT: Weaving the claim web. In Proceedings of 6th International Semantic Web Conference and of the 2nd Asian Semantic Web Conference (ISWC 2007 + ASWC 2007): 197-210. DOI: 10.1007/978-3-540-76298-0_15

[56] Groza, T., Schutz, A., Handschuh, S. (2007). SALT: A Semantic Approach For Generating Document Representations. In Proceedings of the 2007 ACM symposium on Document Engineering (DocEng 2007): 171-173. DOI: 10.1145/1284420.1284462

[57] Harold, E. R. (2004). XML 1.1 Bible (Vol. 136). John Wiley & Sons.

[58] Hickson, I. (2011). HTML5: A vocabulary and associated APIs for HTML and XHTML. W3C Working Draft 25 May 2011. World Wide Web Consortium. http://www.w3.org/TR/html5/ (last visited November 26, 2014).

[59] Hilbert, M., Schonefeld, O., Witt, A. (2005). Making CONCUR work. In Extreme Markup Languages.

[60] Hillesund, T. (2002). Many Outputs Many Inputs: XML for Publishers and E-book Designers. In Journal of Digital Information, 3(1), January 2002.

[61] Horrocks, I., Patel-Schneider, P.F., McGuinness, D.L., Welty, C.A. (2007). OWL: A description logic based ontology language for the Semantic Web. In F. Baader, D. Calvanese, D.L. McGuinness, D. Nardi, P.F. Patel-Schneider (Eds.), The description logic handbook: Theory, implementation and applications, 2nd ed. (pp. 458–486). Cambridge, UK: Cambridge University Press.

[62] Huitfeldt, C., Sperberg-McQueen, C. M. (2001). TexMECS: An experimental markup meta-language for complex documents. http://mlcd.blackmesatech.com/mlcd/2003/Papers/texmecs.html

[63] ISO 5776:1983, Graphic Technology - Symbols for Text Correction. International Organization for Standardization, Geneva, Switzerland. (1983).

[64] Jagadish, H. V., Lakshmanan, L. V., Scannapieco, M., Srivastava, D., Wiwatwattana, N. (2004). Colorful XML: one hierarchy isn't enough. In Proceedings of the 2004 ACM SIGMOD international conference on Management of data. (pp. 251-262). ACM. Doi:10.1145/1007568.1007598.

[65] Jain, A. K., Yu, B. (1998). Document representation and its application to page decomposition. In IEEE Transactions on Pattern Analysis and Machine Intelligence, 20 (3):294.308. DOI: 10.1109/34.667886

[66] Johnson, B., Shneiderman, B. (1991, October). Tree-maps: A space-filling approach to the visualization of hierarchical information structures. In Visualiza-

tion, 1991. Visualization'91, Proceedings., IEEE Conference on (pp. 284-291). IEEE.

[67] Kazman, R. (1986). Structuring the text of the Oxford English Dictionary through finite state transduction. University of Waterloo.

[68] Keim, D. A., Mansmann, F., Schneidewind, J., Thomas, J., Ziegler, H. (2008). Visual analytics: Scope and challenges (pp. 76-90). Springer Berlin Heidelberg.

[69] Knuth, Donald Ervin (1984), The TeXbook, Volume A of Computers and Typesetting. Massachusetts, USA, Addison-Wesley. ISBN 0-201-13448-9.

[70] Koh, E., Caruso, D., Kerne, A., Gutierrez-Osuna, R. (2007). Elimination of junk document surrogate candidates through pattern recognition. In Proceedings of the 2007 ACM symposium on Document engineering (DocEng07). New York, New York: ACM. DOI: 10.1145/1284420.1284466.

[71] Krotzsch, M., Simancik, F., Horrocks, I. (2011). A description logic primer. Ithaca, NY: Cornell University Library. http://arxiv.org/pdf/1201.4089v1 (last visited November 26, 2014).

[72] Lini, L., Lombardini, D., Paoli, M., Colazzo, D., Sartiani, C. (2001). XTReSy: A Text Retrieval System for XML documents. In Augmenting Comprehension: Digital Tools for the History of Ideas.

[73] Liu, A. (20014) Transcendental Data: Toward a Cultural History and Aesthetics of the New Encoded Discourse. Critical Inquiry, 1(31):49–84, 2004.

[74] Marinelli, P., Vitali, F., Zacchiroli, S. (2008). Towards the unification of formats for overlapping markup. In New Review of Hypermedia and Multimedia 14, 1, pages 57-94. doi:10.1080/13614560802316145

[75] McCallum, A., Li, W. (2003, May). Early results for named entity recognition with conditional random fields, feature induction and web-enhanced lexicons. In Proceedings of the seventh conference on Natural language learning at HLT-NAACL 2003-Volume 4 (pp. 188-191). Association for Computational Linguistics.

[76] McGuffin, M. J., Robert, J. M. (2010). Quantifying the space-efficiency of 2D graphical representations of trees. Information Visualization, 9(2), 115-140.

[77] Mendes, P. N., Jakob, M., García-Silva, A., Bizer, C. (2011, September). DBpedia spotlight: shedding light on the web of documents. In Proceedings of the 7th International Conference on Semantic Systems (pp. 1-8). ACM.

[78] Minkov, E., Wang, R. C., Cohen, W. W. (2005, October). Extracting personal names from email: applying named entity recognition to informal text. In Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing (pp. 443-450). Association for Computational Linguistics.

[79] Motik, B., Patel-Schneider, P.F., Parsia, B. (2012). OWL 2 web ontology language: Structural specification and functional-style syntax, 2nd ed. W3C Recommendation, 11 December 2012. World Wide Web Consortium. http://www.w3.org/TR/owl2-syntax/ (last visited November 26, 2014).

[80] Ossanna, J. F. (1980). NROFF/TROFF user's manual. Bell Laboratories.

[81] Pardi, W. J. (1999). XML in Action. Microsoft Press, Redmond, WA, 1999.

[82] Peroni, S., Gangemi, A., Vitali, F. (2011). Dealing with markup semantics. In Proceedings the 7th International Conference on Semantic Systems (I-

SEMANTICS 2011): 111–118. New York, New York, US: ACM Press. DOI: 10.1145/2063518.2063533

[83] Peroni, S., Poggi, F., Vitali, F. (2013). Tracking changes through EARMARK: a theoretical perspective and an implementation. In G. Barabucci, U. Burghoff, A. Di Iorio, S. Maier (Eds.), Proceedings of 1st International Workshop on (Document) Changes: modeling, detection, storage and visualization (DChanges 2013), CEUR Workshop Proceedings 1008. Aachen, Germany: CEUR-WS.org. Retrieved from http://ceur-ws.org/Vol-1008/paper6.pdf (last visited November 26, 2014)

[84] Piez, W. (2001). Beyond the 'descriptive vs. procedural' distinction. In Proceedings of the Extreme Markup Conference, Montreal, Canada, 2001.

[85] Piez, W. (2005). Format and Content: Should they be separated? Can they be?: With a counter-example. In Proceedings of the Extreme Markup Conference, Montreal, Canada, 2005.

[86] Prud'hommeaux, E., Carothers, G. (2013). Turtle - Terse RDF Triple Language. W3C Candidate Recommendation, 19 February 2013. World Wide Web Consortium. Retrieved from http://www.w3.org/TR/turtle/

[87] Quin, L., 1996. Suggestive Markup: Explicit Relationships in Descriptive and Prescriptive DTDs. In GCA SGML Conference, Boston.

[88] Raggett, D., Le Hors, A., Jacobs, I. (1999). HTML 4.01 Specification. W3C recommendation, 24.

[89] Renear, A. H. (2001). The Descriptive/Procedural Distinction is Flawed. Markup Languages: Theory and Practice, 4(2):411–420, 2001.

[90] Schonefeld, O. (2007). XCONCUR and XCONCUR-CL: A constraint-based approach for the validation of concurrent markup. In Data Structures for Linguistic Resources and Applications. Proceedings of the Biennial GLDV Conference 2007, Tübingen, Germany, 2007.

[91] Schulz, H. (2011). Treevis. net: A tree visualization reference. Computer Graphics and Applications, IEEE, 31(6), 11-15.

[92] Schulz, H., Hadlak, S., Schumann, H. (2011). The design space of implicit hierarchy visualization: A survey. Visualization and Computer Graphics, IEEE Transactions on, 17(4), 393-411.

[93] Sekine, S. (1998, May). NYU: Description of the Japanese NE system used for MET-2. In Proc. Message Understanding Conference.

[94] Settles, B. (2004, August). Biomedical named entity recognition using conditional random fields and rich feature sets. In Proceedings of the International Joint Workshop on Natural Language Processing in Biomedicine and its Applications (pp. 104-107). Association for Computational Linguistics.

[95] Shafer, K. E. (1996). Fred: the SGML grammar builder. OCLC, 1996.

[96] Shearer, R., Motik, B., Horrocks, I. (2008). HermiT: A Highly-Efficient OWL Reasoner. In OWLED (Vol. 432).

[97] Shneiderman, B. (1996, September). The eyes have it: A task by data type taxonomy for information visualizations. In Visual Languages, 1996. Proceedings., IEEE Symposium on (pp. 336-343). IEEE.

[98] Shneiderman, B., Wattenberg, M. (2001, October). Ordered treemap layouts.

In Information Visualization, IEEE Symposium on (pp. 73-73). IEEE Computer Society.

[99] Shotton, D. (2009). Semantic Publishing: the coming revolution in scientific journal publishing. Learned Publishing, 22 (2): 85-94. DOI: 10.1087/2009202.

[100] Shotton, D., Portwin, K., Klyne, G., Miles, A. (2009). Adventures in Semantic Publishing: Exemplar Semantic Enhancements of a Research Article. PLoS Computational Biology, 5 (4): e1000361. DOI: 10.1371/journal.pcbi.1000361.

[101] Simpson, P. (1935). Proof-Reading in the Sixteenth, Seventeenth and Eighteenth Centuries (1st ed.). London, Oxford University Press.

[102] Sirin, E., Parsia, B., Grau, B. C., Kalyanpur, A., Katz, Y. (2007). Pellet: A practical owl-dl reasoner. Web Semantics: science, services and agents on the World Wide Web, 5(2), 51-53.

[103] Snooks and Co. (2002). Style manual: for authors, editors and printers (6th ed.). Brisbane, Wiley Australia, 2002.

[104] Spengler, S., Gallinari, P. (2010). Document structure meets page layout: loopy random fields for web news content extraction. In Proceedings of the 10th ACM symposium on Document engineering (DocEng 10): 151-160. DOI: 10.1145/1860559.1860590

[105] Sperberg-McQueen, C. M., Burnard, L. (1997). A Gentle Introduction to SGML. In Guidelines for Electronic Text Encoding and Interchange, pages 13–36, 1997.

[106] Sperberg-McQueen, C. M., Burnard, L. (2000). A Gentle Introduction to XML. In Guidelines for Electronic Text Encoding and Interchange, 2000.

[107] Sperberg-McQueen, C. M., Huitfeldt, C. (1999). Concurrent document hierarchies in MECS and SGML. In Literary and Linguistic Computing, 14(1), 29-42.

[108] Sperberg-McQueen, C. M., Huitfeldt, C. (2004). Goddag: A data structure for overlapping hierarchies. In Digital Documents: Systems and Principles (pp. 139-160). Springer Berlin Heidelberg. doi:10.1007/978-3-540-39916-2_12.

[109] St. Laurent, S. (1997). XML: a primer. IDG Books Worldwide, Inc.

[110] Stasko, J., Zhang, E. (2000). Focus+ context display and navigation techniques for enhancing radial, space-filling hierarchy visualizations. In Information Visualization, 2000. InfoVis 2000. IEEE Symposium on (pp. 57-65). IEEE.

[111] Swales, J.M. (1990). Genre analysis: English in academic and research settings. Cambridge University Press.

[112] Tannier, X., Girardot, J.,Mathieu, M. (2005). Classifying XML tags through "reading contexts". In Proceedings of the 2005 ACM symposium on Document engineering (DocEng05). New York, New York: ACM. DOI: 10.1145/1096601.1096638.

[113] TEI Consortium. (2008). TEI P5: Guidelines for electronic text encoding and interchange. L. Burnard, S. Bauman (Eds.). TEI Consortium.

[114] Tennison, J., Piez, W. (2002). The Layered Markup and Annotation Language (LMNL). In Extreme Markup Languages, 2002.

[115] Thomas, J. J., Cook, K. A. (Eds.). (2005). Illuminating the path: The research and development agenda for visual analytics. IEEE Computer Society Press.

[116] Thuy, P. T. T., Lee, Y. K., Lee, S. (2009). DTD2OWL: automatic transforming XML documents into OWL ontology. In Proceedings of the 2nd International Conference on Interaction Sciences: Information Technology, Culture and Human (pp. 125-131). ACM.

[117] Vion-Dury, J. Y. (2013). Using RDFS/OWL to ease semantic integration of structured documents. In Proceedings of the 2013 ACM symposium on Document engineering (pp. 189-192). ACM.

[118] Vitali, F., Di Iorio, A., Campori, E.V. (2004). Rule-based structural analysis of web pages. In Proceedings of the 6th International Workshop on Document Analysis Systems (DAS 2004): 425.437. Berlin, Germany:Springer. DOI: 10.1007/978-3-540-28640-0_40

[119] Walsh, N. (2002). XML: One Input Many Outputs: a response to Hillesund. Journal of Digital Information, 3(1), January 2002.

[120] Walsh, N. (2008). The DocBook Schema Version 5.0.

[121] Walsh, N. (2010). DocBook 5: The Definitive Guide. Sebastopol, CA, USA: O'Really Media. Version 1.0.3. ISBN: 0596805029.

[122] Witt, A. (2004). Multiple hierarchies: new aspects of an old solution. In Extreme Markup Languages, 2004.

[123] Zhang, L. (2012). Grasping the structure of journal article: Utilizing the functions of information units. Journal of American Society for Information Science and Technology, 63(3), 469–480.

[124] Zou, J., Le, D., Thoma, G. R. (2007). Structure and content analysis for HTML medical articles: a hidden Markov model approach. In Proceedings of

the 2007 ACM symposium on Document engineering (DocEng 07): 199-201. DOI: 10.1145/1284420.1284468